Noname manuscript No.
(will be inserted by the editor)

Effective Deadlock Resolution with Self-Interested Partially-Rational Agents

Nadav Sofy · David Sarne

Received: date / Accepted: date

Abstract This paper pertains to distributed deadlock resolution in settings populated with self-interested bounded-rational autonomous agents. In particular it reports the results of extensive experimentation with 66 agents, each using a deadlock resolution strategy developed and maintained throughout the experiment by a different human decision maker. While from the game-theoretic perspective a simple equilibrium-based solution can be engineered for the problem, it is shown that such a solution fails to hold with bounded-rational agents, even when its principles are thoroughly explained to the individuals that maintain the agents' strategies. Instead, we show that the system converges to a steady-state, in which the agents use a rich set of different strategies, varying in their performance, as each of them has a different belief regarding its ability to improve, based on the behaviors of the others. To improve system performance, we develop and implement a restructuring heuristic that changes the input each agent receives, as a means of affecting the agents' decisions to better align with the desired solution. The restructured deadlock presented to each agent is based on former deadlocks it had experienced. Our experiments demonstrate the effectiveness of the restructuring heuristic in facilitating a new steady-state in which the system as a whole substantially improves its performance. The efficiency of the method, in terms of the size of the set of former experiences required for effectively restructuring the agents' input, is demonstrated through a comparison with a neural network implementation of input restructuring, showing a substantial advantage to the restructuring heuristic.

Keywords Multi-agent systems · Distributed deadlock resolution · Bounded-rational agents

1 Introduction

In many multi-agent environments, both physical and virtual, deadlocks may occur. These deadlocks usually take the form of a cyclic set of requests each agent makes to the next

Nadav Sofy
Bar-Ilan University, Ramat-Gan 52900 Israel E-mail: nadavsof@gmail.com

David Sarne
Bar-Ilan University, Ramat-Gan 52900 Israel E-mail: sarned@cs.biu.ac.il

agent in the cycle [62,27,32], holding its regular course of task execution. The deadlock can be resolved only if one of the agents chooses to opt-out of the deadlock, i.e., waves off its requests from others, and fulfills requests from others (e.g., releases the resources it holds, without first receiving the resources it requests).

The deadlock problem occurs in many different contexts [30]. For instance, in peace talks between two warring nations if both parties refuse to give up any occupied land while demanding the return of some land held by their adversary, the talks could be blocked indefinitely. Another common example is the traffic deadlock where four cars arrive at a four-way intersection at approximately the same time, each wishing to proceed ahead or turn left. If they all obey the right-of-way regulations, none of them will be able to proceed, indefinitely. Finally, deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a software lock or soft lock [28]. As such, the problem has received much attention in Operating Systems and Databases literature [55,52,2,42], resulting in various mechanisms for avoiding, detecting and recovering from deadlock situations [6,36]. Yet, most of the research in this area focuses on centrally managed systems, which assume that agents are inherently cooperative and can be preempted (e.g., can be forced to release their resources), as needed, by the system manager (e.g., the Operating System) [36]. Recent advances in deadlock research have extended the deadlock model to distributed environments where deadlocks are more difficult to manage since none of the participating agents has full knowledge concerning the system's state. Consequently, a number of approaches have been pursued for deadlocks avoidance [37,6, 32] and detection [34,10] in distributed systems. Still, all the methods designed for deadlock resolution in distributed settings assume that agents are cooperative and follow a dictated deadlock resolution protocol [51,41,55,47,35].

In many deadlock situations occurring in multi-agent systems agents are, however, self-interested (e.g., in negotiation or traffic problems), and a cooperative deadlock resolution scheme cannot be enforced. This situation, of deadlocks with self-interested agents, is also likely to hold in future virtual open environments where agents migrate between different platforms, communicating and negotiating with other agents, autonomously, without the mediation of hosting operating systems and platforms. In such environments, deadlocks can be resolved only if an agent willingly retracts from its deadlock-related requirements (e.g., releases the resources it holds or cancels its requirements from the other agents). The problem becomes even more complex if the agents are not fully rational or pre-programmed with different deadlock handling logics (e.g., programmed by different users). In this case, each individual agent needs to be incentivized to comply with the required behavior that will lead the system to an effective deadlock resolution. The vast deadlock management solutions discussed in literature, based on the assumption that agents are cooperative and fully rational, may become irrelevant in such settings.

In this paper we report the results of extensive experimentation with deadlock resolution in distributed settings of self-interested, partially-rational agents. Each agent's deadlock resolution strategy, in our experiment, was programmed and continuously maintained by a different individual, corresponding to situations where different programs or agents running on the platform are programmed by different programmers or software companies.[1] The goal of the agents was to finish executing their task as early as possible. The essence of the deadlock resolution strategy was therefore deciding, based on the deadlock parameters and the time elapsed since it was formed, whether or not to opt-out. While a decision to opt-out

[1] Meaning that the agents do not use self-adapting strategies or any kind of learning but rather the changes in their strategies are fully attributed to the will of their human software developers.

from the deadlock early in the process guarantees a quick deadlock resolution, the agent opting-out would need to wait until all the other agents in the deadlock finish executing, before it could continue to execute its own task. On the other hand, holding the decision to opt-out, hoping that the deadlock will be resolved by others has the risk of waiting a substantial time in the deadlock until it is resolved.

The paper makes several contributions with respect to deadlock resolution in distributed settings of the type described above. First, it provides evidence of the ability of such systems to reach a steady-state. Naturally, when an agent repeatedly operates in the same environment, its designer may attempt to improve its deadlock management strategy by better adapting to the environment. We show that such adaptation processes, even when simultaneously performed by all users along time, result in the eventual (relatively quickly) convergence of the different strategies to a steady-state, where none of the participants is interested in further changing her agent's strategy. The convergence to a steady-state is demonstrated in all the settings that were examined.

Second, evidence is given of the failure of game-theoretic-based approaches in the distributed deadlock resolution domain, where agents' strategies are set by rationally-bounded (human) designers. The resulting steady-state strategies in our experiments are not efficient system-wise, as the average waiting time of the agents in deadlocks is relatively substantial in comparison to the theoretical bound. Game theory principles can be applied to construct a simple stable distributed solution to the problem which guarantees immediate, and more important, mostly effective deadlock resolution. Yet, prior literature has suggested that game-theoretic-based solutions often do not hold where people or rationally-bounded agents are concerned [4,40,64,7]. Our attempt to suggest such a solution resulted in poor performance of our testing environment, even though the solution was thoroughly explained to the individuals that maintain the agents' strategies. Once the testing environment reached a steady-state, none of the strategies used by the agents followed the game-theoretic solution.

Finally, as an alternative to the game-theoretic approach, the paper demonstrates the usefulness of input restructuring as a means of improving the system performance in settings such as those discussed in the paper. In particular, the paper introduces an adaptive heuristic for restructuring, i.e., changing the deadlock description the agents receive as an input. Meaning that for the same deadlock encountered by a group of agents, each agent may receive a different deadlock as an input for its decision making process. This new approach parallels recent developments in psychology, behavioral economics and agents' design [58, 31,50], suggesting restructuring (i.e., changing) the decision-making problem itself rather than attempting to directly change a decision-maker's strategy. The idea is that the decision maker, believing that the restructured deadlock is indeed the deadlock faced, will act different than when facing the actual deadlock. The heuristic thus attempts to present to each agent a different (restructured) deadlock that is likely to affect its decision to opt-out in a way that better aligns with the desired behavior. This way, deadlocks can be resolved more efficiently system-wise. The restructuring heuristic was tested in the same experimental environment. Its effectiveness derives from its ability to lead the system to a new steady-state in which the system as a whole substantially improves its performance as demonstrated in our experiment. Its efficiency is demonstrated through a comparison with a legacy neural network implementation of input restructuring, showing that the proposed restructuring heuristic requires a substantially smaller database of prior observations than the latter to reflect the same performance improvement.

All in all, the results reported in this paper mainly apply to system designers, highlighting the inherent inefficiency of unintervened distributed deadlock resolution when the decision makers are not fully rational. Much of this inefficiency can be substantially overcome

using a relatively simple to employ and manage restructuring heuristic such as the one suggested in this paper. Indeed, in systems that guarantee data integrity, the approach of input restructuring is not feasible. However, as our experimental results demonstrate, whenever the system is allowed to provide a restructured input to agents, the improvement achieved (both system-wise and individually) is substantial.

The paper is organized as follows. In the following section we introduce the formal deadlock model. Section 3 provides a detailed theoretical analysis of the model, agent-wise and system-wise, introducing the different performance bounds that are used later to analyze the results. It also gives the game-theoretic solution to the problem. Section 4 presents the proposed input restructuring heuristic, designed to improve the system performance in such settings. Section 5 details the experimental methodology used to evaluate the convergence to a steady-state and the effectiveness of the input restructuring heuristic. Section 6 presents the results and their analysis. Section 7 describes three complementary experiments and their results. The first supplies a better understanding of the sources of improvements achieved with the input restructuring heuristic. The second aims to evaluate the usefulness of supplying the decision makers with the appropriate input regarding the deadlock existence and its parameters in the first place. The third demonstrates how the restructuring heuristic can be adjusted to balance between individual and system performance. Finally, in Section 8 we review related literature and we conclude with a discussion and directions for future research in Section 9.

2 The Deadlock Model

The model considers a distributed multi-agent system, where agents are self-interested, partially-rational and possibly represent different users. Each agent is assigned a task, for which it needs several different (reusable) resources to complete it. The goal of each agent is to minimize the time it takes to complete its task. The modeling of deadlocks in the system fully aligns with the standard Coffman deadlock model, commonly found in literature on Operating Systems [52, 12]. According to this model, resources first need to be requested (to be acquired) by an agent and once used are released.[2] The system is considered to be in a deadlock state if a circular chain $(A_1, ..., A_N, A_1)$ of agents (the "circular wait condition") exists, where each agent A_i attempts to acquire a resource held by agent A_{i+1} (A_1, in case $i = N$) in order to proceed with its execution (the " hold and wait condition"). This can typically be represented by a resource allocation graph or a wait-for graph [52] (see Figure 1). It is assumed that a resource cannot be used by more than one agent at a time (the "mutual exclusion condition"), and that a resource cannot be forcibly taken from an agent holding it but can only be released by an explicit action of the agent (the "no preemption condition").

Each agent A_i is associated with a time t_{A_i}, representing the time it is planning to use the resource it is waiting for (the resource currently held by A_{i+1}) after having acquired it. Therefore, t_{A_i} is also the time it takes A_i to release the resource A_{i-1} is waiting for from the time it attains the resource for which it is waiting. The deadlock can thus be represented as $A = ((A_1, t_{A_1}), ..., (A_N, t_{A_N}), (A_1, t_{A_1}))$. If an agent willingly releases the resource it holds (opt-out) it will need to acquire both the resource released (now held by the former agent in the chain) and the resource it was waiting for (held by the next agent in the chain) in order to proceed with its task. We assume these two resources are acquired as soon as they

[2] While the terminology used in the literature on Operating Systems considers processes as the system's blocked entities, we prefer agents due to the context of self-interested entities.

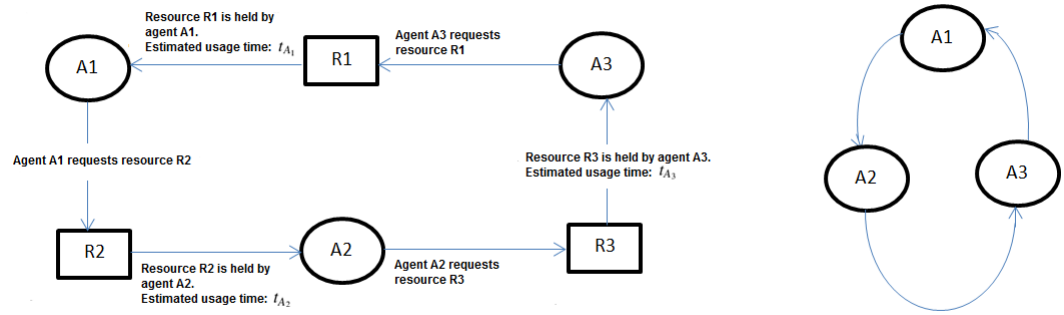

Fig. 1 Resource Allocation Graph (left) and its corresponding Wait-For Graph (right) of a circular deadlock state. Agents are represented by circles and resources by rectangles. A_1 is holding resource R_1, and is waiting for A_2 to release resource R_2. Once A_1 receives resource R_2 it will need t_{A_1} time units to complete its task. A_2 will not release resource R_2 until it acquires resource R_3 and finishes its task, which is estimated to take t_{A_2} time units. Meanwhile, A_3 is holding resource R_3 and is waiting for resource R_1 and thus the system is in a deadlock.

are released by the agents currently holding them. For example, in Figure 1, if A_1 releases resource R_1 that it holds after t' time units, then A_3 immediately gains hold of R_1 and executes the task for t_{A_3} time units. A_2 waits t_{A_3} time units until attaining R_3 and then executes the task for t_{A_2} time units. The last to execute is A_1, which opted-out and resolved the deadlock. After waiting $t_{A_3} + t_{A_2}$ time units for both A_3 and A_2 to complete their tasks, A_1 will re-acquire R_1 back and R_2, and will then execute the task for t_{A_1} time units. The waiting times of the different agents in this example are: t' for A_3 which executed the task immediately after A_1 opted-out, $t' + t_{A_3}$ for A_2 and $t' + t_{A_3} + t_{A_2}$ for A_1.

We assume the existence of a central entity (e.g., an operating system), that supplies system-related information to the agents, which cannot preempt the agents' hold of resources or enforce any particular agent behavior. In particular, since agents in a multi-agent system (MAS) can block their regular operation for various reasons (i.e., not necessarily due to a deadlock), we assume the system informs the agents once they are actually in a deadlock and supplies them with the deadlock description.[3] This latter information consists of a vector of times, where the i-th element in the vector represents the time associated with the i-th agent along the circular chain (where the agent receiving the information is the first in the chain).[4] In Figure 1, for example, the deadlock descriptions A_1, A_2 and A_3 receive are $(t_{A_1}, t_{A_2}, t_{A_3})$, $(t_{A_2}, t_{A_3}, t_{A_1})$ and $(t_{A_3}, t_{A_1}, t_{A_2})$, respectively.

An agent's strategy, when in a deadlock A, is thus the mapping $S : A \mapsto t$, where t is the time since the deadlock is first reported to the agent until the agent willingly opts-out (if the deadlock is not resolved by another agent by then). As mentioned above, each agent's goal is to minimize the time it takes to complete its task. Since no agent has control over its own processing time, this goal is equivalent to minimizing its overall waiting time in the

[3] Deadlock detection is supported by various mechanisms [10, 34] which can be done in $O(n^2)$ operations, where n is the number of vertices in the wait-for graph.

[4] It is assumed that the times t_{A_i} $(i \leq n)$ are known to (or can be learned by) the central entity. For example, the central entity can estimate the time it takes agent A_i to complete a specific task according to statistics over previous executions of this task, or alternatively, the agents truthfully disclose this value (where truthfulness is enforced by the system through the denial of future service or other means).

deadlock (i.e., the time elapsed since the deadlock was formed until the agent managed to acquire the resources it requested and began executing the task). From the system's point of view, we consider the sum of utilities realized by the agents (utilitarian social welfare) as common in multi-agent settings [44], i.e., the goal is to minimize the expected average waiting time of the agents.

3 Analysis

In this section we analyze the agents' and the system's performance, and develop the performance bounds that are used later to analyze the experimental results. For exposition purposes, we use $A_{sub}(A_i, A_j)$ to represent the subchain of the agents positioned between A_i and A_j (exclusive) in deadlock A. We also assume, WLOG, that the deadlock is formed at time $t = 0$.

Once an agent $A_i \in A$ opts-out from a deadlock, the deadlock is resolved. In this case agent A_i will need to wait a time $\sum_{A_k \in A \wedge k \neq i} t_{A_k}$, while any other agent A_j will wait a time equal to the total processing times of all the agents along the subchain $A_{sub}(A_j, A_i)$ (formally, given by $\sum_{A_k \in A_{sub}(A_j, A_i)} t_{A_k}$). Notice that once the deadlock is resolved by agent A_i, no agent $A_j \neq A_i$ can reduce the time it needs to wait by individually opting-out as well. This is because opting-out will not affect the time any of the agents in $A_{sub}(A_j, A_i)$ will need to wait until acquiring the resource they request.

If the deadlock is resolved at time $t' > 0$, then the agent that resolved the deadlock could have done better had it opted-out at time $t = 0$. In fact, all the agents in the deadlock would have reduced their waiting time by t' had the agent resolving the deadlock opted-out at time $t = 0$ rather than $t = t'$. Still, always opting-out at time $t = 0$ is definitely not the best strategy for a self-interested agent. Using the $t = 0$ strategy when none of the other agents are using it results in always waiting for all the other agents to finish executing their task, hence waiting $\sum_{A_k \in A \wedge k \neq i} t_{A_k}$ time units. If all the agents use a $t = 0$ strategy then any single agent would improve its performance by switching to a $t > 0$ strategy. Using a $t \to \infty$ strategy (i.e., holding onto the resource and never choosing to opt-out) is beneficial if the deadlock is resolved relatively early by one of the other agents. However, if all agents use this strategy then they will all wait indefinitely.

Let $F_{(A,i)}(t)$ denote the probability that a random agent placed as the i-th agent in deadlock A will opt-out at or before time t. Assuming t is continuous, the derivative $f_{(A,i)}(t) = \frac{dF_{(A,i)}(t)}{dt}$ is the probability density function (PDF) of having the i-th agent in deadlock A opt-out at time t. The optimal (i.e., expected-waiting-time-minimizing) strategy t^* when being the j-th agent in the deadlock is thus given by:[5]

$$t^* = argmin_t \Big(\int_{y=0}^{t} \Big(\sum_{A_k \in A, k \neq j} f_{(A,k)}(y) \prod_{w \neq j,k} (1 - F_{(A,w)}(y)) \cdot \Big(y + \sum_{A_l \in A_{sub}(A_j, A_k)} t_{A_l} \Big) \Big) dy + \Big(\prod_{w \neq j} (1 - F_{(A,w)}(t)) \Big) \Big(t + \sum_{k \neq i} t_{A_k} \Big) \Big) \tag{1}$$

The right hand side of the equation calculates the expected waiting time of the j-th agent for any opt-out time t it can possibly use and returns the one resulting in the minimum value. The first part of the term relates to cases where the deadlock is resolved by one of

[5] A similar equation can be formulated for the case of discrete t values, substituting the integral with a sum, and $f_{(A,i)}(t)$ with $P = F_{(A,i)}(t+1) - F_{(A,i)}(t)$, and handling ties.

the other agents at any time $y < t$. The calculation iterates over all other agents, taking the probability each of them will be the first to opt out at time t, in which case agent A_j waits $y + \sum_{A_l \in A_{sub}(A_j,A_k)} t_{A_l}$. The second part relates to the case where A_j is the first to opt-out, i.e., when all other agents choose to opt-out later than t, thus it waits t plus the sum of all other agents' time. The time t^* that minimizes the expression is therefore the best opt-out strategy for agent A_j when in deadlock A. Yet, the use of such a strategy requires a huge database of results, to accurately extract the probability function $F_{(A,i)}(t)$, given the many possible combinations of processing times, as well as the number of agents in the deadlock and each agent's position in the deadlock.

From the system's point of view, the optimum (i.e., expected-average-waiting-time-minimizing) deadlock resolution is achieved when the deadlock is resolved at $t' = 0$. This is because any solution according to which the agent to opt-out first opts-out at time $t' > 0$ is dominated by the same solution, except that the agent opts-out at time $t' = 0$. If the deadlock is resolved by agent $A_i \in A$ at time $t' = 0$, then the total waiting time is the sum of waiting times of each agent $A_{j \neq i}$ plus the time agent A_i itself has to wait until all other agents finish executing their task. Thus, the average waiting time, is given by:

$$\frac{1}{N}\Big(\sum_{j \neq i} \sum_{A_k \in A_{sub}(A_j,A_i)} t_{A_k} + \sum_{k \neq i} t_{A_k}\Big) \tag{2}$$

Therefore from the system's point of view, a lower bound for the expected average waiting time is obtained when agent A_i, for which Equation 2 is minimized, opts-out at time $t' = 0$. Taking the setting given in Figure 1 as an example, if the processing times are $t_{A_1} = 6, t_{A_2} = 3$ and $t_{A_3} = 5$, then the best outcome is achieved if A_3 opts-out at time $t' = 0$, resulting in a total waiting time of 12 time units (A_2 executes its task immediately, A_1 waits 3 time units and A_3 waits a total of 9). It is noteworthy that even though A_1 has the longest processing time, if it opts-out at time $t' = 0$, the resulting total waiting time is 13 (A_3 executes its task immediately, A_2 waits 5 time units and A_1 waits a total of 8 time units).

Theoretically, having agent A_i, for which the average waiting time according to Equation 2 is minimized, opt-out at $t' = 0$, can be achieved in the distributed setting this paper considers using game theory principles. For example, given a deadlock A, the agents can follow a strategy according to which only the agent that minimizes Equation 2 opts-out at time $t' = 0$ and all the remaining agents never choose to opt-out (i.e., set $t' \to \infty$). This strategy is in equilibrium, since none of the agents has an incentive to deviate from it: as all other agents use $t' \to \infty$, the agent that is supposed to opt-out at $t' = 0$ will adhere to this strategy. As discussed earlier, any other strategy $t > 0$ is dominated by $t = 0$ for this agent, and using $t \to \infty$ will result in an overall infinite waiting time. As for the other agents, since the deadlock is supposed to be resolved at time $t = 0$, none of them will have an incentive to deviate to a different strategy. In fact, based on the same argument, any protocol whereby one of the agents is selected to opt-out at time $t < \infty$ according to some pre-defined criteria (e.g., having the minimum or maximum processing time) while all other agents use $t \to \infty$ is in equilibrium.

The theoretical analysis given in this section can be used to compare the performance of any two sets of agents' strategies from the system's point of view. The measure uses the following supplementary notations:

- B, D - two sets of agents' strategies.
- $Performance_B$, $Performance_D$ - the average waiting time (over all agents) with sets B and D, respectively.

– $TheoreticalPerformance$ - the theoretical minimum average waiting time (according to Equation 2).

The difference between $Performance_B$ or $Performance_D$ and $TheoreticalPerformance$ is the overhead (or "inefficiency") due to the use of the non-optimal set of strategies in environments B and D, respectively. The performance improvement of the system, when switching from environment B to environment D, measured as the change (in percentages) in the overhead time, is given by:

$$1 - \frac{Performance_D - TheoreticalPerformance}{Performance_B - TheoreticalPerformance} \tag{3}$$

This choice of using the overhead as a the main measure for improvement is because the overhead is the component out the overall waiting time over which the restructuring heuristic can actually affect (as a waiting time lower than the theoretical performance is unattainable). Any other measure (e.g., taking the overall expected waiting time) would highly depend on the deadlock settings generated for the experiments.

While Equation 3 is used throughout this paper to compare the system performance under different treatments, we are also interested in comparing individual performance of agents under these treatments. The measure for individual performance change makes use of the overhead change concept as above, defining:

– $Performance_B(A_i)$, $Performance_D(A_i)$ - the average waiting time of agent A_i when operating alongside agents of sets B and D, respectively.[6]
– $TheoreticalAgentPerformance$ - a lower bound for the expected average waiting time of an agent.

The difference between $Performance_B(A_i)$ or $Performance_D(A_i)$ and the bound $TheoreticalAgentPerformance$ is the overhead (or inefficiency) due to the use of a non-optimal set of strategies in environments B and D, respectively. The individual performance improvement of agent A_i's performance when switching from environment B to environment D, measured as the change (in percentages) in the overhead time, is given by:

$$1 - \frac{Performance_D(A_i) - TheoreticalAgentPerformance}{Performance_B(A_i) - TheoreticalAgentPerformance} \tag{4}$$

We note that unlike in the system's performance measure, the individual agent's theoretical minimum expected waiting time depends on the set of the other agents in the system. Therefore, the measure uses a lower bound for the agent's individual performance, $TheoreticalAgentPerformance$, which does not depend on the strategies used by the other agents. This bound, which is explained in more detail in 5.1, essentially considers the case where the deadlock is always resolved at time zero (therefore it does not depend on the other agents' strategies) and the agent is never the first to opt out. The improvement we report in individual performance, whenever relating to that measure, is therefore a lower bound for the actual improvement.

4 Restructuring Heuristic

While the game-theoretic-based solution can be theoretically engineered to guarantee the best possible system performance, as discussed in the former section, there is much empirical evidence of failure of such mechanisms in settings where the agents are not fully rational

[6] Notice that the strategy of A_i itself may change in the transition from B to D.

[59,19], e.g., when programmed by people [40,9]. In this paper, we use an alternative approach, a heuristic that restructures the deadlock-related information the agents receive as input from the system. The idea is to indirectly influence the agents' deadlock-related decisions in a way that improves the overall system performance. Applying such an approach requires the agents to be unaware of the fact that the input they receive is being restructured, in order to prevent attempts of acting strategically, which might decrease the effectiveness of the input restructuring approach. This latter approach builds on recent developments in psychology and behavioral economics [58,31] as well as in multi-agent systems [50,4,25].

In the distributed deadlock resolution domain the problem restructuring approach offers various advantages in comparison to attempting to directly influence the decision-maker's strategy so that it aligns better with the desired one. First, since the agents are self-interested it is very difficult to incentivize each individual agent to act in a desired manner. Second, even if the appropriate incentives can be constructed, the direct approach often requires argumentation, since the bounded-rational agents may fail to realize the benefits in the proposed solution. The restructuring approach, on the other hand, completely avoids the need to persuade the decision-maker of the usefulness or benefits of the system-desired strategy. Finally, in most cases the strategy of an agent is pre-set and cannot be changed externally, unless it is re-programmed, and thus the agent simply cannot be persuaded by the system. In such cases, only the restructuring approach can influence the agent's decisions. Despite its many advantages, the new approach should be carefully managed, as the reaction of agents to the restructured information may not always align with the system designer's expectations. Since the agents' strategies are not a priori known, it is possible that some of the agents will not act as expected, and the use of the restructured input will end up making things worse. Furthermore, based on the performance of the agents with the restructured data, the agents' designers/owners may decide to re-design their agents' strategies and the system may end up doing worse than with the original agents' strategies. Also, as discussed in the introduction, this new approach is not applicable whenever the system is obligated or regulated to supply a non-restructured deadlock, e.g., when the system's code is offered as an open source (e.g., as in Linux). Still, whenever input restructuring is applicable, the individual and overall benefits are substantially improved, as demonstrated in the following sections.

Unlike prior work where the input restructuring approach is used [4,58,31], our use of input restructuring attempts to affect the collective behavior of a group of agents rather than a single agent, by supplying a revised input (deadlock description) to each agent separately. The goal of effectively influencing the collective behavior of a group of agents is more complex in our case, as any agent that does not respond as expected, might negatively affect the whole group. Another difference is that in the deadlock setting, the system and agents' individual goals often conflict, whereas in prior work the focus has been solely on the benefit of the agent whose input is restructured [50].

Our use of the restructuring approach for distributed deadlock resolution suggests that each agent in the deadlock be presented a variation of a deadlock it has previously encountered rather than the deadlock it is currently encountering. The alternative deadlock presented to the agent may differ from the original deadlock in any parameter (e.g., number of other agents and their processing times). The only restructuring constraint in our case is the true processing time of the agent itself which is kept as is (as this latter information is known to the agent or can be later verified). The deadlock variation eventually used is thus a previously encountered deadlock in which the agent is either most likely to wait the longest time until opting-out or the one for which the agent is most likely to wait the shortest time until opting-out (depending on the role planned for that agent). The restructuring heuristic

modifies the selected deadlock by changing the agent's own time to its time in the current deadlock.

Algorithm 1 Adaptive Restructurer

Input: $A^{current}$ - the current deadlock the system is in; DB - Set of prior deadlock instances; $MinRestructured$, α - the minimum number and percentage of agents to attempt to influence to opt-out first, respectively;
Output: $(A^1, ..., A^{|A^{current}|})$ - the tuple of restructured deadlocks to be presented to the agents.
// Extract local DB for each agent A_i
Set $DB_i = \{A | A \in DB \wedge A_i \in A\}, \forall 1 \leq i \leq |A^{current}|$
Set $Explore = null$; $Exploit = null$
// Exploration with probability $1/(|DB_i|)^2$, otherwise exploitation
for $A_i \in A^{current}$ **do**
  if $rand() < 1/(|DB_i|)^2$ **then**
    Add A_i to $Explore$
  else
    Add A_i to $Exploit$
  end if
end for
// Identify potential deadlocks to be presented to each exploited agent and evaluate performance
for $A_i \in Exploit$ **do**
  Set $MinA_i = \arg\min_A(A.resolutionTime | A \in DB_i \wedge A.resolverAgent = A_i)$
  Set $MaxA_i = \arg\max_A(A.resolutionTime | A \in DB_i)$
  Set $MaxA_i[A_i].processingTime = MinA_i[A_i].processingTime = A^{current}[A_i].processingTime$
  // Calculate the expected performance when agent A_i is the first to opt-out
  Set $Performance_i = \left(\sum_{j \neq i} \sum_{A_k \in A^{current}_{sub}(A_j, A_i)} t_{A_k} + \sum_{k \neq i} t_{A_k}\right) + MinA_i.resolutionTime$
end for
// Remove infeasible solutions
Move the set $\{A_i | A_i \in Exploit \wedge MinA_i.resolutionTime > \min\{MaxA_j.resolutionTime | j \neq i\}\}$ from $Exploit$ to $Explore$
Order $Exploit$ according to $Performance_i$
Set $k = max(\lceil \alpha | A^{current} | \rceil, MinRestructured)$
Set $A^i = MinA_i$ for the top k agents in $Exploit$ and $A^i = MaxA_i$ for any other $A_i \in Exploit$
Set $A^i = A^{current}$ for any $A_i \in Explore$
return $(A^1, ..., A^{|A^{current}|})$

The restructuring heuristic is described in Algorithm 1. Since the heuristic relies on former deadlocks the agents encountered, it uses a classical exploration versus exploitation approach, with a decaying exploration [3,5]. The heuristic is based on three main phases: (a) exploration versus exploitation decision; (b) predicting worse and best opt-out times given the current database of formerly observed deadlocks, for each agent, that can be achieved using input restructuring; and (c) deciding on the restructured deadlock each agent receives as input.

The heuristic receives the current deadlock description the system is in, which includes the identities of the agents and their positions in the deadlock (denoted by "$A^{current}$"), and the set of formerly observed deadlocks (denoted by "DB"). Each formerly observed deadlock in DB has the identity of all agents that did not receive it in its original masked form (i.e., in its non-restructured form). This is because the behaviors of these agents in that deadlock are correlated with the restructured deadlocks they received rather than the deadlock stored in the database. Both $A^{current}$ and DB are made available by the system using

the heuristic. After the resolution of each deadlock, the system adds the deadlock to DB, masking all agents that were not presented with that deadlock, as explained above. Restructured deadlocks are not stored in DB in order to prevent situations where the exploitation is based on a set of rather homogeneous deadlocks.[7] The heuristic outputs a set of deadlocks to be presented to the agents in the current deadlock. Based on the set of formerly observed deadlocks, DB, the heuristic first extracts (as a preprocessing step) the relevant deadlocks for each agent A_i, denoted DB_i (Step 1). The relevant deadlocks are those that were actually presented to the agent. The heuristic then executes its exploration versus exploitation phase (Steps 2-9). In this phase, it is decided for each agent whether it will be given the current deadlock $A^{current}$ as input (i.e., exploration), in an attempt to enlarge the set of experiences associated with that agent, or a modification of one of the deadlocks it has already encountered, attempting to exploit the knowledge the system has on this agent. The decision between exploration and exploitation (Step 4) is made for each agent separately, based on the number of observations available for this agent (i.e., the size of DB_i). Agents for which exploration needs to take place are added to the $Explore$ list and the others to the $Exploit$ list. The probability of choosing to exploit (presenting a restructured deadlock to the agent) is $(|DB_i|^2 - 1)/|DB_i|^2$, and the probability of choosing to explore (presenting $A^{current}$) is $1/|DB_i|^2$ (coinciding with the "least taken" multi-armed bandit algorithm [60], a variant of the ϵ-decreasing approach, that results in a faster decrease of the exploration probability), where $|DB_i|$ is the number of different deadlocks stored in the system for that agent. While we attempt to keep this step simple, several improvements can be considered when determining exploration and exploitation. For example, exploration can be avoided completely for an agent for which a deadlock in which that agent opts-out at time $t = 0$ with certainty was found. Another potential improvement can be obtained by adjusting the probability to explore or exploit not only based on the number of examples in DB_i but also on the quality of those examples (e.g., if the earliest time an agent was recorded to opt out is still relatively high, more exploration needs to be done).

In the second phase (Steps 10-16), the heuristic attempts to predict its potential influence over the agents that are designated to receive a restructured deadlock (i.e., for every agent in the list $Exploit$) in terms of the earliest and latest times they are likely to opt-out with restructured inputs. For the earliest time the agent will opt-out, it considers all the deadlocks in DB_i where that agent was the first to opt-out (Step 11). Here, $A.resolverAgent$ denotes the identity of the agent that opted-out in deadlock A and $A.resolutionTime$ denotes the time at which the agent opted-out. For the latest time, it considers all the deadlocks in DB_i, since the time the agent would have waited in these deadlocks until opting-out is inevitably greater or equal to the deadlock's resolution time (Step 12). The deadlock in which agent A_i opted-out earliest and the one in which agent A_i opted-out (or was still holding onto its resource) the latest are stored in $MinA_i$ and $MaxA_i$, respectively. The processing time of each agent in its restructured deadlock (both for the minimum and latest opt-out time) is overridden by the actual processing time according to $A^{current}$ (Step 13). This is required since the restructured input is constrained by the requirement to present to each agent its own true processing time as discussed above. $MaxA_i[A_i].processingTime$, $MinA_i[A_i].processingTime$ and $A^{current}[A_i].processingTime$ denote the processing time of agent A_i in the deadlocks $MaxA_i$, $MinA_i$ and $A^{current}$, respectively (Step 13).

Based on its prediction of the level of influence it can have over each agent's decision to opt-out, the heuristic chooses the agents to be influenced to opt-out first. The overall waiting

[7] The homogeneity property derives from the fact that the revised deadlocks are similar to the existing deadlocks they are based on.

time when A_i is the first to opt-out, according to its $MinA_i$ input, is calculated (based on Equation 2) for every $A_i \in Exploit$ and stored in $Performance_i$ (Step 14). This enables the system to predict the expected system performance if it successfully manages to influence A_i to opt-out first. Agents that cannot be influenced to opt-out first, i.e., their predicted opt-out time if given the $MinA_i$ restructured deadlock is greater than at least one agent A_j's predicted opt-out time according to its $MaxA_j$, are removed from the $Exploit$ list. This is because there is no point in trying to influence those agents to opt-out first. Instead, the heuristic uses them for exploration (Step 16).

Finally, in the last phase (Steps 17-21), the heuristic chooses the k agents from $Exploit$ that yield the lowest overall waiting times if they opt-out first, given the deadlock $A^{current}$. The reason for attempting to influence more than a single agent to opt-out first through input restructuring is that there is no guarantee of successfully influencing the agent associated with the best $Performance_i$ (as even though the deadlocks given to the agents are based on deadlocks retrieved from DB, these deadlocks are different in the values assigned to the agents' individual processing time). The value of k is therefore determined as a percentage of the number of agents in the deadlock, α, while guaranteeing a pre-defined minimum of $MinRestructured$ (Step 18). Each of the k agents that are associated with the k-best $Performance_i$ values in the $Exploit$ list are therefore given the restructured deadlock according to which it is likely to opt-out as early as possible (Step 19). The remaining agents in $Exploit$ receive the restructured deadlock according to which they are likely to opt-out as late as possible (Step 19). The agents in $Explore$ receive the current deadlock $A^{current}$ unchanged in order to enrich their set of experiences in DB with the new (current) deadlock (Step 20). The set of deadlocks to be presented to each agent is returned as an output (Step 21).

We note that while the approach used by Algorithm 1 attempts to maximize performance system-wise, it does not address fairness issues. In particular, the attempts to influence agents to align with the desired behavior, system-wise, may not be evenly spread across the agents, resulting in having some agents suffer a degradation in their individual performance. Designing a more balanced solution requires minor changes, mostly in terms of keeping track of the accumulated effect of the restructuring heuristic on the different agents and incorporating this information in the decision regarding which agents to try to influence to opt-out first. This is thoroughly discussed in Section 7.3.

One important condition for the success of the restructuring heuristic presented in this section, is the ability to identify the agents. This capability is needed in order to correlate agents with the set of observations representing former deadlock experiences (Step 1). This is often possible in real distributed systems, as agents can be identified by the system through the use of cookies, as well as other diverse enforceable identification methods, and certainly in operating systems where security and authentication mechanisms are widely applied [21].

5 Evaluation

The goals set for the evaluation are threefold: First, we wanted to provide evidence of the ability of a system where deadlocks are resolved distributively by self-interested bounded-rational agents to eventually reach a steady-state. A steady-state, within this framework, is one where none of the users believes in her ability to change her agent's strategy in a way that will improve performance, given the behavior of other agents. This definition of a steady-state in a way resembles the notion of a Satisficing equilibria [56], which is a state in which all agents reach an outcome that is sub-optimal, but is also an acceptable one. Agent

designers in a system that is in a Satisficing equilibria are reluctant to change their agent's strategy because they do not know, or do not have the available resources to find a better strategy and are satisfied with the performance of their current strategy. This definition of a steady-state is slightly different than the definition of a Nash equilibrium steady-state, in which none of the users can objectively improve the performance of her agent given the strategies used by others. The difference between the two also suggests that in a steady-state of the first type we are likely to observe various different strategies, each yielding different expected waiting times when in a given deadlock, due to the inherent differences between different individuals and their beliefs. On the other hand, in the case of the Nash equilibrium for the problem (any game-theoretic solution of the type discussed above), the symmetry of the agents' roles in the different deadlocks suggest that all agents end up with a similar average waiting time. The second evaluation goal was to test our hypothesis that the game-theoretic-based approach will not hold in the distributed deadlock resolution domain, where agents are rationally-bounded. Finally, we wanted to demonstrate that the system performance can be substantially improved using the input restructuring approach, and test the effectiveness of the proposed restructuring heuristic as a possible means of reducing individual and average waiting times. As a part of this latter goal, we aimed at demonstrating the convergence of the system to a steady-state when the restructuring heuristic is applied and comparing the proposed heuristic with an alternative implementation of input restructuring (neural-network-based in this case).

5.1 Simulation Infrastructure

For these purposes, we developed a system that simulates Coffman deadlocks[8] in an operating system, where the processes are self-interested agents. The agents in the system are put in a deadlock upon instantiation. They receive the deadlock description as input, which includes the processing time of every agent in the deadlock, and the order of the agents in the chain. Each agent used by the system was programmed by a different person (henceforth denoted "strategy designer") based on a generic skeleton, that was only missing its strategy layer. This separation principle in the design and the use of strategy designers is common practice in multi-agent research [50, 11, 16]. The only functionality the strategy designers needed to implement was the decision of whether to opt-out from the deadlock based on the deadlock description and the time elapsed (in terms of clock ticks) from the time the deadlock was formed. Deadlocks were generated automatically and assigned with agents, randomly, allowing repetition.[9] Upon initialization, each agent received the deadlock description. At each time step, the agent was requested to decide whether to opt-out from the deadlock or not. Once an agent opted-out, the system terminated and the average waiting time was calculated according to Equation 2. Due to the discrete nature of the system (which is tick-based), if two or more agents opted-out at the same time step, then one of them was randomly chosen to be the first to opt-out. A timeout of several orders of magnitude greater than the mean processing time was used, in case none of the agents was willing to opt-out

[8] A Coffman deadlock [12] is a deadlock satisfying the four conditions outlined at the beginning of Section 2: Mutual Exclusion, Hold and Wait, No Preemption and Circular Wait. Unfulfillment of any of these conditions is enough to preclude a deadlock from occurring.

[9] We allowed repetition in order to simulate two processes that were instantiated based on the same program running concurrently (or alternatively, a process created based on a fork() system call of another process). When two instances of the same agent participated in the same deadlock, no communication was allowed between them. Furthermore, neither of the agents was aware of the existence of two agents that implement the same strategy.

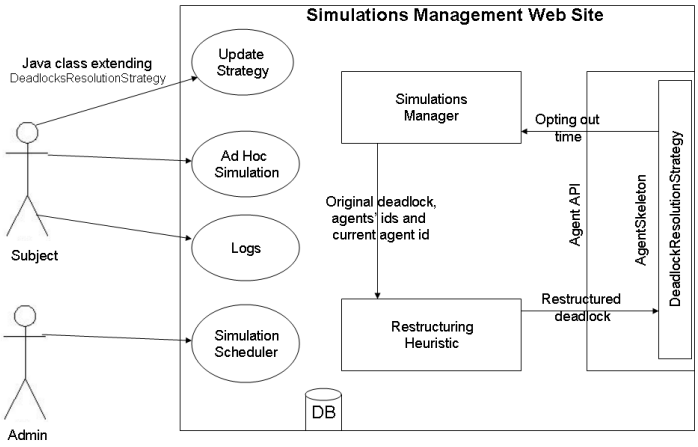

Fig. 2 Evaluation System's Architecture. The system includes three modules providing the required functionality for the agents' designers: Update Strategy for submitting modified agent strategies, Ad Hoc Simulation for testing the agent's performance with the latest version of the other agents and the Logging module, which displays historical data of deadlocks in which the user's agent participated. The Simulation Scheduler provides the functionality of running evaluation sessions. The Simulations Manager is used for generating random deadlocks and assigning agents to them . The architecture of an agent consists of a skeleton which implements all the required functionality, except for the opting-out strategy, which is designed and programmed by the strategy designers.

after that time, forcing (randomly) one of the agents to opt-out. The value of the timeout was known to the strategy designers. It is noteworthy that this timeout was never reached along the experiment as all strategies in all stages had implemented their own, substantially lower, timeouts.

The evaluation system included an advanced web-based interface and its architecture is presented in Figure 2. The main purpose of this interface was to enable users to update the agents' strategies along the different stages of the experiment (Update Strategy module) and test an agent's strategy performance based on the most current version of the other agents in the system (Ad Hoc Simulation module). In addition, the interface included a logging module ("Logs"), that enabled the strategy designers, at any stage of the experiment, to access historical simulation results of their agents (including the deadlocks' descriptions) and filter those by different versions of their agent's strategy. The core of the evaluation system is the simulation manager that is responsible for generating random deadlocks (simulating an agents' platform which detects deadlocks) and assigning them with agents. A designated restructuring heuristic module received the simulated deadlock description and the IDs of the agents in it, and restructured the deadlock description each agent received according to Algorithm 1. In stages where the restructuring heuristic was not applied, the simulation manager presented the generated deadlocks to the agents directly. Finally, a Simulation Scheduler module enabled us to provide schedules of the dates available for simulations to the strategy designers, enabling them to prepare for the simulations in advance, as well as run ad-hoc, unscheduled simulations.

The size of a new generated deadlock (i.e., the number of agents participating in it) was randomly drawn from the range 2-10 to reflect a large-scale system, where hundreds of agents execute concurrently, as considered in prior deadlock studies [63, 14]. The processing times were drawn from an Erlang distribution (a specific case of hyper exponential distribu-

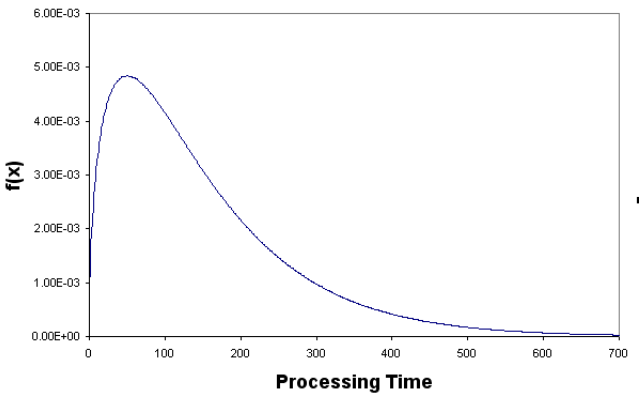

Fig. 3 Processing time distribution - Erlang distribution with parameters $\lambda = 0.01$ and $k = 1.5$. The mean is 150. This distribution qualitatively resembles the typical distribution of CPU burst times in operating systems.

tion, where $f(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$) with parameters $\lambda = 0.01$ and $k = 1.5$ (See Figure 3). The mean value of this distribution is 150. The Erlang distribution was chosen because it qualitatively resembles the typical distribution of CPU burst times in operating systems [52]. Both the distribution of processing times, and the range from which the number of agents in the deadlock is drawn were given to the strategy designers as part of the experiment's instructions, and were fixed throughout all the experimental stages.

Based on the deadlock parameter values detailed above, several important theoretical performance measures can be calculated and several observations can be made:

- Individual performance measures:
  - The expected waiting time of an agent A_i, when in a deadlock of size s, given that another agent A_j opts-out of the deadlock immediately is $\sum_{k=0}^{s-2} \frac{150k}{s} = \frac{150(s-2)(s-1)}{2s}$ since the average processing time of each agent is 150 and the probability of having k agents in $A_{sub}(A_i, A_j)$ is $\frac{1}{s}$, as that agent can be anywhere in the deadlock chain with an equal likelihood. The probability of being in a deadlock of size $2 \leq s \leq 10$ is $\frac{1}{9}$ as the deadlock size is drawn (uniformly) from [2,10]. The expected waiting time of an agent that never opts-out first while all the deadlocks it participates in are resolved immediately, is therefore $\sum_{s=2}^{10} \frac{1}{9} \sum_{k=0}^{s-2} \frac{150k}{s} = 257$. This scenario is the most optimistic from a single agent's point of view thus the value 257 is used later as the theoretical lower bound of the average waiting time of an agent.
  - The expected waiting time of an agent which opts-out first, when in a deadlock of size s is $150(s-1)$, as 150 is the mean individual processing time. The probability of being in a deadlock of size s is $\frac{1}{9}$ as detailed above. The expected waiting time of an agent that is always the first to opt-out is therefore $\sum_{s=2}^{10} \frac{150}{9}(s-1) = 750$. This value is used later as the zero score of a strategy, as it can trivially be achieved by immediately opting-out from every deadlock.
  - The performance of an agent which never opts-out is severely affected by its results in deadlocks where the other agents use the same strategy or are instances of that same agent, and is therefore in the order of magnitude of the timeout set for deadlocks.

– From the system's point of view, the average waiting time if the agents obey the optimal deadlock resolution scheme (according to Equation 2) is 366. This value was obtained empirically based on more than one million simulations whereby the agent for which the average waiting time is minimized was forced to opt-out.[10] This value is a lower theoretical bound of the system performance, and is in fact what one would expect the game-theoretic approach to the problem would achieve had the solution been adopted by all the strategy designers.

These bounds are used later to compare both individual and system performance in the different experimental stages.

5.2 Methodology

In order to make the evaluation as realistic as possible, the experimentation was carried out over a few weeks, allowing the strategy designers to adapt their strategies over time, based on the results of thousands of deadlocks in which their agents participated.[11] In particular, each strategy designer received a timely report, based on a set of scheduled (and from time to time, ad-hoc unscheduled) system evaluations. Each evaluation included 5000 deadlocks of the form described above with the agents assigned to each deadlock randomly chosen. A report summarizing each evaluation was sent to each strategy designer, and included deadlocks in which her agent participated. Each deadlock record included the description of the deadlock (the processing times of the agents in the deadlock, though without disclosing the identity of the other agents), its resolution time and the position of the agent which was the first to opt-out. In addition to the individual performance of each agent on each evaluation, the system also stored the system's average performance (agents' average waiting time).
The experimentation included four stages (see the experiment's timeline in Figure 4). The following paragraphs give the definition of and methodology used for each stage of the experiment.

Stage 1 - Unknown Environment At this stage, participants were introduced to the formal deadlock model, emphasizing the individual performance calculation principle as outlined in Section 3. After receiving a detailed explanation regarding the agent development task, they were requested to devise their own agent's initial deadlock management strategy. The set of strategies received at this stage, thus, applies to the case of deadlocks in an environment which is completely new to all agents. These agents were used for the first evaluation of individual and system performance.

Stage 2 - Convergence to a steady-state At this stage, each strategy designer received the results of the evaluation of the agent which strategy she designed in the previous stage. The strategy designers were requested to continuously review the performance of their agents (in terms of average waiting time) and update their strategies accordingly. This stage was facilitated by the system interface, that enabled participants not only to review former deadlocks the agent participated in, but also to generate more deadlocks and to test the performance of different strategies with other agents that were randomly selected by the system (without

[10] In this case, the extensive experimentation replaces the direct calculation due to the complexity of the latter.

[11] As in reality, a programmer is likely to change the deadlock resolution logic of her software, based on data collected through many experiences in a given computer system.

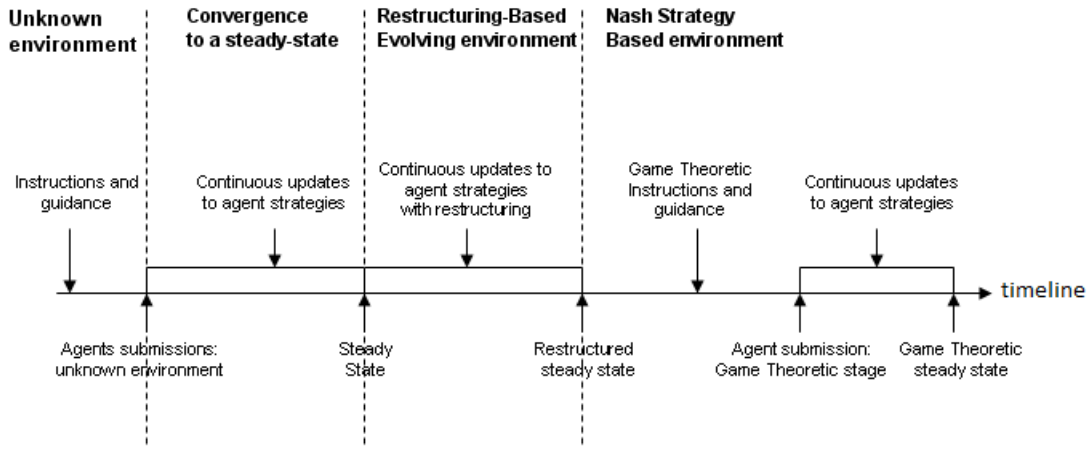

Fig. 4 Experimental stages along the timeline. The first stage simulated an environment that is new ("unknown") to all agents. The second stage consisted of continuous updates of agent strategies, until converging to a steady-state whereby none of the strategy designers were interested in applying changes to their agent's strategy. At this point of the experiment, the restructuring heuristic was applied, leading to a new sequence of updates to the agent strategies until converging to a new steady-state. Before the game-theoretic stage the system was reset and a detailed explanation of the game theory principles relevant to this stage was given to the strategy designers as well as a suggested Nash equilibrium strategy for the deadlock resolution problem. This stage, as the two stages before it, simulated an evolving system and ended in a new, game-theoretic-based, steady-state.

disclosing their identity). The report each strategy designer received after each evaluation can be considered the equivalent to logs that users can generate in real-life systems. The main purpose of this stage was to simulate a scenario where agents' strategies evolve over time, in response to changes in the strategy used by the other agents in the environment.

This process of repeated strategy updates and evaluations of the agents' performance continued until one week without any change made to any of the agents' strategies elapsed. We chose this stopping criterion because the time it takes to update an agent's strategy is meaningless compared to a week. We also stress that while all strategy designers were students at the time of the experiment, as described in the following paragraphs, their curriculum varied, and therefore it is very unlikely that an external event made all participants stop updating their agent's strategy in the same week. The set of agents stored in the system at the end of this stage was considered the steady-state strategies and was used as input for the next stage of the experiment.

Stage 3 - Restructuring-Based Evolving Environment The goal of this stage was to allow the agents' strategies to evolve with the restructuring mechanism affecting their decisions, until the system reached a steady-state once again. For this purpose, the restructuring heuristic was first applied over the steady-state strategies yielding different performances for each agent in comparison to its performance in the previous stage.[12] The use of the restructuring heuristic was transparent to the strategy designers, and as far as they were concerned, the

[12] The heuristic was used with the values $MinRestructured = 2$ and $\alpha = 0.5$ in order to ensure that at least one of the agents that is influenced to opt-out as early as possible will be the one which will opt-out first.

change in their agent's performance was attributed to changes in other agents' strategies as in former evaluations.

The strategy designers continued to review their agent's results and update their strategies accordingly, while the evaluation was taking place with input restructuring. As in the preceding stage, the strategy designers were able to continuously run their agent in deadlocks with other agents in the environment and change its strategy based on its performance. This process of repeated strategy updates and evaluations of performance again continued until one week without any change made to any of the agents' strategies elapsed. The set of agents stored in the system at the end of this stage was considered the restructured steady-state strategies.

Stage 4 - Nash Strategy Based Environment Finally, the strategy designers were given a detailed explanation about the game-theoretic-based solution to the problem. For this purpose we used the Nash equilibrium according to which the agent with the longest processing time in each deadlock would be the one to opt-out. Although this is not the dominating Nash equilibrium from the systems point of view, this choice was made in order to simplify coding and to assure that a deviation from the equilibrium strategy would not be the result of buggy implementations or difficulty in understanding the strategy.[13] The expected waiting time of this suggested equilibrium is 406, obtained empirically by means of one million simulations due to the complexity of the calculation.[14] The drawbacks of deviating from this strategy, assuming all other agents use it, were fully discussed and detailed in the task description. The use of this strategy was suggested to the strategy designers though it was made clear to them that there is no centralized mechanism enforcing it (i.e., this strategy is optional, and up to each strategy designer to decide whether or not to use it).

The strategy designers were requested to program a new strategy for this stage (though they could have re-submitted any of their strategies from earlier stages) to encourage the participants to implement the Nash strategy and to ensure that if the strategy designers would rely on their strategy from the preceding stage it would not be without considering the Nash strategy. The agents were evaluated without the restructuring heuristic, and the convergence process was managed as in the two prior stages of the experiment (i.e., continuous strategy updates and evaluations of performance until one week passes without any strategy updates made).

5.3 Strategy Designers and Scoring Incentives

The strategy designers were computer science students in a core Operating Systems course. This group fairly represents future agent developers who are likely to design the deadlock resolving logic for distributed systems of the type considered in this paper. The experimentation took place after the students became formally familiar with the deadlock concept as part of the regular course curriculum.

The design of the agent deadlock resolution strategy was offered as a bonus (volunteered) assignment. Those students who were willing to accept the challenge, were requested to submit an agent based on the skeleton described above, with the goal of minimizing its

[13] The exact implementation (which is practically three lines of pseudo code) was supplied to the students.

[14] Regardless of the average waiting time guaranteed by the equilibrium, if a strategy designer believes all the other agents will adopt the equilibrium strategy then so should she, as a deviation from the equilibrium strategy would result in performance degradation.

expected waiting time given the deadlock parameters detailed above. The grade for the assignment was fully correlated with the average performance of the agent throughout the different stages of the experiment. Only students who participated in the first stage of the experiment (the unknown environment) were allowed to continue and participate in the subsequent stages. Since the most time-consuming part of the assignment was to understand the setting and to write the initial strategy, students found it worthy to keep updating their strategy with the belief that the change may lead to a performance improvement in upcoming runs. The maximum scoring was set to the theoretical lower bound (257, which is practically impossible to achieve) and a zero score was received if the performance of the agent was equal or below the performance of an agent that is always the first to opt-up at time $t = 0$ (750). We chose this latter strategy's expected waiting time as the zero score as it is the most naive strategy one can contemplate, is trivial to implement, and does not reflect any attempt to perform well in the deadlock resolution domain. An average waiting time between these two values received a score relative to its position in the interval (e.g., an average of 503.5 received 50). Participants had to provide documentation describing their deadlock management logic and were requested not to share any information between each other since their performance might be affected from doing so. Overall, 66 students chose to participate in the experiment.

5.4 Handling Noise and Outliers

As a matter of principle, all agents receiving a zero score when using their steady-state strategy (a total of 19 agents) were tagged as "dummy" agents. These were mostly agents that opted-out of any deadlock immediately, or agents that for some deadlock descriptions waited for a very long time until opting-out. The first exhibit poor performance because they were always the last to execute their task. The latter exhibit poor performance because when participating in a deadlock where all other agents use a similar strategy, the deadlock is resolved after a very long time, causing their performance to degrade substantially. The natural instinct was to completely remove these agents from the system (e.g., in consecutive stages), since seemingly there is no explanation for why a strategy designer would not try to change a strategy yielding a zero score. Yet, such agents are part of any real-life system (e.g., zombie processes or non-critical processes where their programmer is not concerned with their deadlock management logic). Even in physical environments it is not rare to find people who are always the first to back out of a deadlock, even as a matter of courtesy. Therefore, these agents were kept as part of the pool of agents used by the system. Nonetheless, as a means of precaution, their results were filtered from the analysis. We emphasize that by including the results of these agents in the analysis given below, far better results are obtained. However, these results would not reflect a correct analysis of the system performance improvement, as a system administrator would rather improve the performance of agents whose designers actually greatly care about how they perform.[15]

[15] The reasons for the great improvement achieved with these agents are that they are easy to identify, and once identified, their behavior can be accurately predicted. If there are cases (even rare) where these agents do choose to opt-out early (for those that usually wait a long time) or late (for those that usually opt-out immediately) then the improvement achieved by restructuring their input is substantial. In fact, including the dummy agents in the evaluation yields a relative improvement of 83% in system performance in the restructured steady-state in comparison to the non-restructured steady-state, compared to a 35% relative improvement (as reported in Section 6.3) when excluding the dummy agents from the analysis.

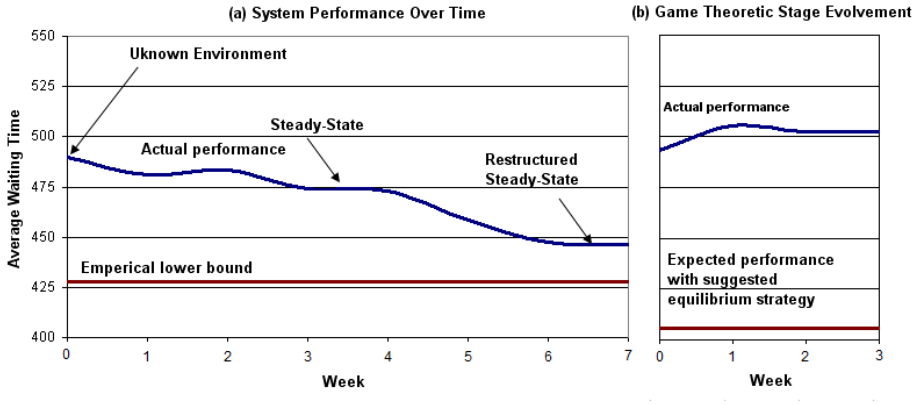

Fig. 5 System performance evolvement over the experimental stages, aligned to actual timeline (in weeks). The steady-state, restructured steady-state and game-theoretic steady-state performance are characterized by a flat curve as none of the agents' strategies was changed for seven days once these were reached.

6 Results Analysis

Figure 5 aligns the different experimental stages along the experiment's actual timeline, depicting the system performance throughout each stage. From the time all individual strategies were first constructed (for the unknown environment), 4 weeks elapsed until a steady-state was reached. Similarly, from the time the restructuring heuristic was first applied, 3 weeks elapsed until a new steady-state was reached. When starting from scratch, after suggesting the game-theoretic strategy to the subjects, 3 weeks elapsed until a steady-state was reached. We re-emphasize that a steady-state in our case does not mean a state where no individual agent can improve its performance, but rather a state of the system in which none of the strategy designers believes that she can improve her strategy.

We first present a general analysis of the types of strategies used by the strategy designers for their agents, and their evolvement over the experimental stages. Then, we review the results obtained by the different classes of strategies along the different stages of the experiment, and finally the effectiveness of the proposed restructuring heuristic. Whenever applicable, we use t-test for determining if the performance achieved with one set of strategies is significantly different from the one achieved with another (e.g., comparing performance with and without the restructuring heuristic), reporting also the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true (i.e., the $p - value$).

6.1 Agent Strategies

The analysis of the strategies used by the strategy designers is based on both the agents' documentation and the review of the code itself. The first interesting observation based on the review of the strategies is that none of the 66 agents that were received used, at any stage of the experiment, a strategy resembling the optimal one which is based on modeling the distribution of the opt-out time per position in a deadlock and act accordingly, as discussed in the analysis section. While such a strategy might not be effective in practice as

it requires a very large set of observations of the environment, this strategy was not even tried at any stage of the experiment. Table 1 details the distribution of non-dummy agent strategies according to a natural parameter-based classification. Since there are 3 parameters that can affect the agents' decision - total processing time (denoted 'Total'), own processing time (denoted 'MyTime') and the size of the deadlock in terms of the number of agents in it (denoted 'Size'), we first distinguish between the 2^3 combinations of these three parameters, each defining a different strategy class. The class in which none of the three parameters is taken into account is further divided into a 'Constant' (e.g., "opt-out at $t = 3$") and 'Random' (e.g., "opt-out at a random time t, where t is drawn from [4,10]") strategy. Two additional strategies which are distinguished in the table relate to the game-theoretic-based strategy. The first (denoted 'Nash'), precisely follows the game-theoretic-based strategy described in Section 3: release only if MyTime is the longest, otherwise hold infinitely. The second (denoted 'Nash with Empty Threat') follows the first part of the strategy (release if MyTime is the longest) however does not entirely follow the second (keep holding infinitely otherwise). Instead, after some pre-defined time, the agent using this strategy variant would choose to opt-out. As can be observed from (their absence in) Table 1, none of the agents, in any of the experiment stages, used strategies relying solely on the agent's own time, or a combination of the agent's own time and the deadlock size. One possible explanation for this is that whenever taking into account the agent's own time, it makes sense to include the total-time as well, basing the decision on how long the agent's own time is relative to the total processing time of a deadlock, rather than basing it on the absolute value of the agent's own time parameter.

Figure 6 depicts the individual performance of the different agents (excluding "dummies"), for the unknown environment and for the steady-state with and without the restructuring heuristic. Since we are interested in the collective behavior rather than individuals', the agents in each graph are ordered according to their performance, thus the i-th agent in each graph does not necessarily represent the same agent. As can be observed from the figure, the performance of different strategies of the same class highly varies, indicating substantial differences in the coefficients and thresholds used by different subjects when basing the decision on the same set of parameters. Based on Figure 6 and Table 1, several additional observations relating to strategy evolvement can be made. First, there is a continuous decrease over time in the use of strategies that rely only on own and total times (denoted 'func(MyTime,Total)'). Though according to the graphs, such strategy has potential to perform well (as it is used by some of the top-ranked agents), a thorough investigation of the

Strategy Class	Unknown	Steady-State	Restructured Steady	Game-Theoretic First	Game-Theoretic Steady
func(Total)	39%	43%	43%	17%	26%
func(Size)	4%	4%	4%	4%	4%
func(Total, Size)	19%	19%	23%	29%	34%
func(MyTime, Total)	25%	14%	11%	4%	6%
func(MyTime, Total, Size)	4%	7%	9%	14%	15%
Constant	7%	11%	8%	10%	9%
Random	2%	2%	2%	4%	3%
Nash	0%	0%	0%	0%	0%
Nash with Empty Threat	0%	0%	0%	18%	3%

Table 1 Distribution of classes of strategies among the non-dummy agents in different experimental stages (a total of 47 non-dummy agents). See details regarding the different classes in the text.

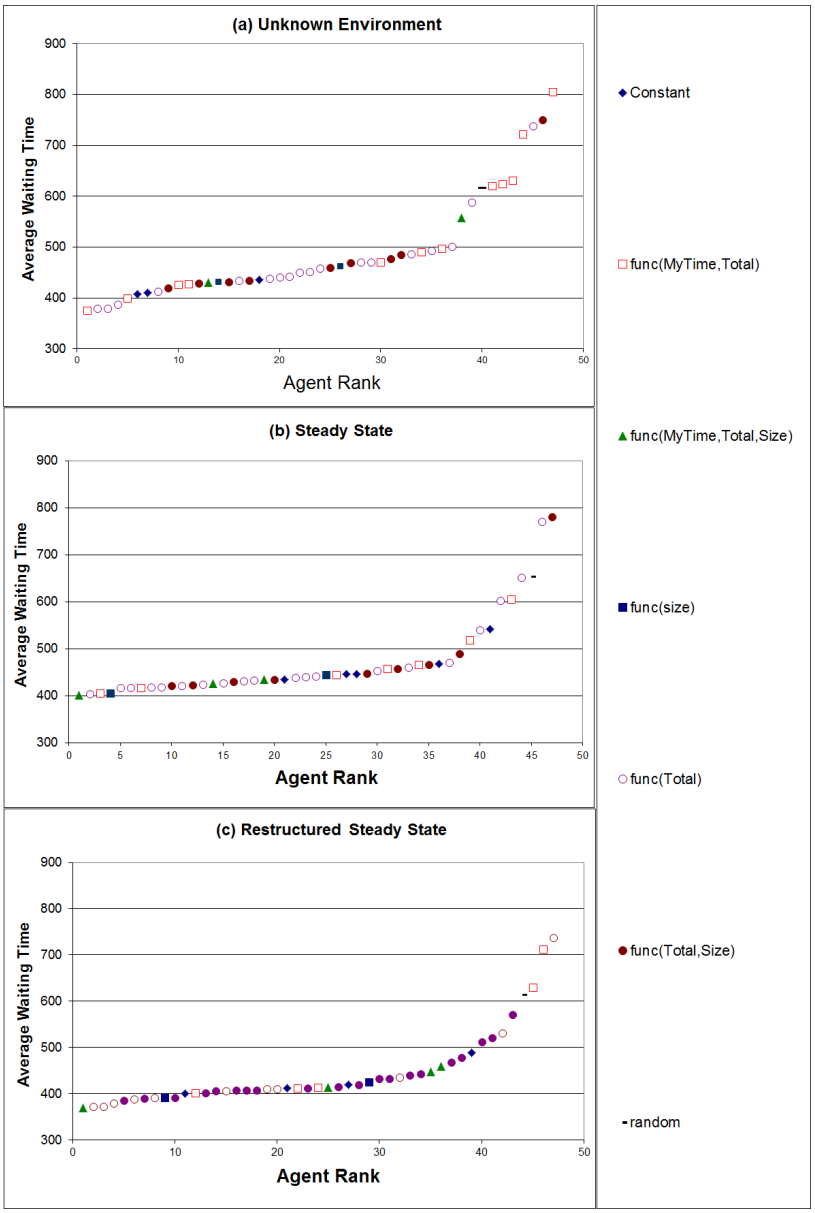

Fig. 6 Performance according to strategy classes in the Unknown, Steady-State and Restructured Steady-State environments. Agents in each graph are ordered according to their performance.

usage patterns of that strategy reveals some interesting insights. Apparently, there are two common implementations for this strategy. The first decides whether to opt-out immediately or to opt-out at a very late constant timeout based on a function of the two parameters (e.g., "if $MyTime$ is greater than 150 and $Total$ is lesser than 400, opt-out immediately, else wait until a large timeout c passes"), while the second decides when to opt-out based on

these same parameters (e.g., "opt-out after 5 time units if $Total$ is lesser than 50, else opt-out at $t = MyTime$"). The resulting average waiting times of all the agents using the latter implementation of the func(MyTime,Total) strategy was substantially smaller than the constant timeouts used by agents using the former implementation of this strategy class. This distinction makes a great difference in the resulting performance: strategies of the first type performed poorly, and thus were replaced by others over time. Strategies of the second type performed initially well and remained competent along time. Overall, the 'Total' parameter was used in 83% of the steady-state strategies, the 'Size' parameter was used in 30% of the steady-state strategies and the 'MyTime' parameter was used in 21% of the steady-state strategies. In the restructuring-based steady-state, the numbers are similar with slightly higher values for total time and size. It is noteworthy that deadlock size and the total time parameters are inherently correlated (the more agents in the deadlock, the more likely it has a greater total-time) whereas there is no correlation at all between the deadlock size and own time parameters when constructing deadlocks. Therefore, the total time seems to also capture the size of the deadlock and thus explaining its importance. Additional evidence of less importance of the agent's own time parameter is the fact that none of the strategy designers used it as the sole parameter for the decision.

One interesting insight obtained from Table 1 is that only one agent out of the 66 used randomization as part of its strategy. The importance of this fact is mostly in supporting our general restructuring approach in the sense that agents are consistent. Meaning that the agents are not using mixed strategies and a deadlock that was found to be useful in pushing an agent to opt-out early or late is likely to achieve the same effect if slightly modified or if used as is once again.

Finally, from reviewing the documentation and code of the agents, we gain two additional insights. First, we see that most agents that used the func(Total,Size) strategy based their opt-out time on the average processing time (i.e., $Total/Size$). Second, we see that the effect of the average processing time on the time the deadlock will be resolved is inconclusive. While most of the agents that based their strategy on the average time opt-out earlier the smaller the average time, a few agents do the exact opposite, i.e., opt-out earlier the greater the average time. We observe a similar behavior in agents who based their strategy only on the total time of the deadlock. Most agents opt-out earlier the smaller the total time is, but some do the opposite. Unlike strategies based on the total time and the deadlock size (which usually rely on the average processing time) or solely on the total time of the deadlock, strategies that use the agent's own time as one of the decision parameters are more consistent, as agents opt-out earlier the greater their own time.

6.2 Convergence to A Steady-State

Figure 7 comparatively depicts the individual performance of the different agents (excluding "dummies"), for the unknown environment and for the steady-state with and without the restructuring heuristic. As in Figure 6, the i-th data point in each curve represents the performance of the i-th best agent at that stage. This figure can explain the convergence and adaptation of different subjects to the deadlock resolution dynamics in the new environment when starting from the unknown environment set of strategies. The convergence is reflected by the "flattening" of the distribution of individual performance (between "unknown environment" and "steady-state"). Figure 8 depicts the change in each of the agents' performance, measured as the percentage of change in the waiting time overhead (in comparison to the agent's individual theoretical lower bound, as defined in Section 5.1) according

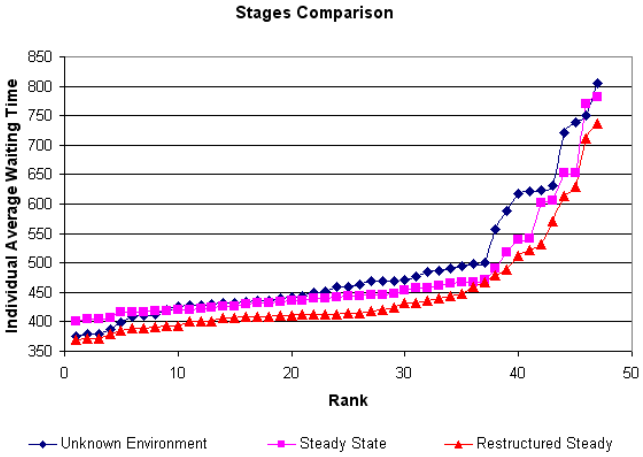

Fig. 7 Comparative performance of the agents in the different experimental stages. The agents are ordered from the agent with the best performance as number one to the one with the worst performance in each stage.

to Equation 4. As can be observed from the figure, some of the agents managed to improve their performance by changing their strategies, at the expense of others (see the discussion on strategy evolvement in the preceding paragraph). Still, from Figure 5(a), we see that the average waiting time of the system actually decreased from an average waiting time of 490 at the beginning of the experiment to an average of 475 at the steady-state (before applying the input restructuring mechanism), reflecting an improvement of 12% in system inefficiency (according to Equation 3). This decrease is not statistically significant ($p = 0.14$ for t-test), and indeed, in one of the other convergence situations reported in this paper the steady-state performance was actually worse than at the beginning of the stage (in the game-theoretic stage). Also, we observe from Figure 5(a) an initial decrease in average waiting time, then an increase and then again a decrease until no more changes were noted. The first decrease is partially attributed to the correction of initial inherent inefficiencies of agents' strategy, while the changes that followed are attributed to adaptation of the strategies to the environment.

6.3 Convergence with the Restructuring Heuristic

From Figure 7 we see that unlike the case of converging from an unknown to a steady environment, where the distribution of the agents' performance practically "flattens", the convergence to a restructuring-based steady-state has different characteristics. Here, the distribution actually "shifts down", improving the performance of most agents. Overall, applying the restructuring heuristic resulted in a steady-state with an average waiting time of 446, compared to 475 in the steady-state when the restructuring is not applied, reflecting an improvement of 35% in the system inefficiency (according to Equation 3). Unlike the decrease in average waiting time from unknown to steady-state, this decrease is statistically significant ($p < 0.001$ for t-test). Another difference from convergence to a steady-state from an unknown environment is that here, the average waiting time constantly decreases over time (see Figure 5(a)). The explanation for this is that at the steady-state, most subjects believed their individual strategy could not be improved further. Therefore, the strategy designers

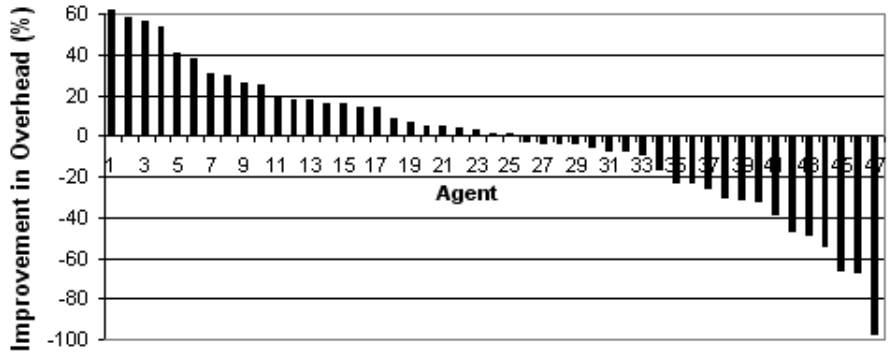

Fig. 8 Performance improvement of each agent in the steady-state environment, relative to its performance in the unknown environment. The agents are ordered from the agent who had the highest relative performance improvement (Agent number 1 in the graph), to the agent that suffered the highest performance decrease in the steady-state.

whose agents' performance improved due to the restructuring heuristic were reluctant to make further changes in their strategy. A total of 12 agents' performance worsened by the restructuring heuristic in the first iteration of this stage and only 5 in a significant manner that reflected a non-negligible decrease in their grade for that iteration. Since the number of agents whose performance worsened after the heuristic was applied was relatively small, the effect of the changes in their strategy on the performance of the other agents was not substantial and thus the other strategy designers did not see any benefit in further revising their strategy, especially given the substantial improvement in their performance in comparison to the steady-state.

One important capability of the restructuring heuristic is its ability to effectively control the agents' opting-out time (either shortening or extending it in comparison to the non-restructured case). This capability is affected by the efficiency of the embedded exploration versus exploitation process, as discussed in Section 4. Figure 9 depicts the average waiting time (system performance) as a function of the number of deadlocks observed until then (each data point is the average of 1000 simulations), when applying the restructuring heuristic on the restructuring based steady-state's strategies. Several observations can be made based on the graph. First, when the number of observations available in the database is relatively small (less than 100), the improvement in the system performance is trivial as the quality of predictions is poor.[16] As more deadlock records accumulate in the database, the improvement becomes more obvious and increases at a greater rate until new observations have little effect on the heuristic's efficiency and the performance slowly converges to 446. We also observe that the main extent of improvement is achieved after 500 iterations, suggesting that the heuristic learns quite effectively which restructured deadlocks are likely to result in the desired behavior of each agent.

To further illustrate the efficiency of the restructuring heuristic, we performed an offline learning process in which each agent was assigned to 1000 random deadlocks for each possible processing time in the range of 1-300 which covers 95% of the distribution (see Figure 3), while all other agents' strategies were set to waiting indefinitely. Doing so enabled us

[16] Note that the number of deadlocks in the database relates to DB, of which DB_i is only a subset. Therefore the number of records the heuristic bases the prediction on for each agent is smaller.

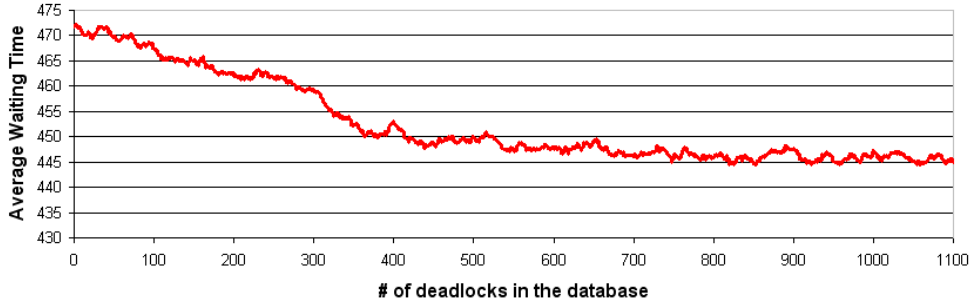

Fig. 9 Average waiting time (system performance) with the restructuring heuristic as a function of the number of deadlocks observed by the system thus far. The set of agents used for this figure is the one representing the restructured steady-state.

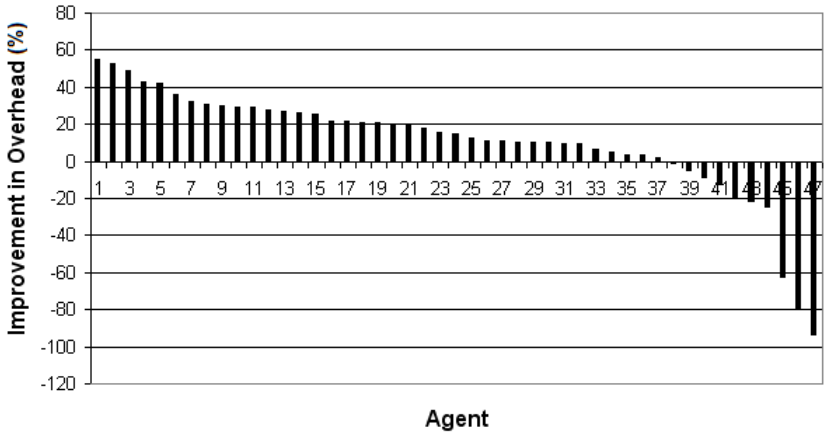

Fig. 10 Performance improvement of each agent in the restructured steady-state environment, relative to its performance in the non-restructured steady-state environment. The agents are ordered from the agent who had the highest relative performance improvement (Agent number 1 in the graph), to the agent that suffered the highest performance decrease in the steady-state with the restructuring heuristic applied.

to find the ultimate opt-out time minimizing and maximizing restructured deadlocks. These deadlocks where used in an offline post-experiment evaluation of the system, yielding an average waiting time of 428 (compared to the theoretical bound of 366). This new value can be considered an experimental bound, as none of the deadlocks can be resolved in a more efficient way given the absolute minimum and maximum waiting times of the different agents strategies (e.g., given the set of agents' strategies, this is how much the system could have been improved by the restructuring heuristic when having near perfect knowledge of the agents' strategies). Taking the experimental bound as a benchmark for calculating the reduction in the (system-wise) waiting time overhead achieved with the restructuring heuristic, the reduction from 475 (in steady-state) to 446 (in restructured steady-state) reflects a reduction (i.e., improvement) of 61% in the overhead.

Figure 10 depicts the change in each of the agents' performance in the restructured steady-state environment, relative to its performance in the non-restructured steady-state environment, measured as the percentage of change in the waiting time overhead (in compari-

son to the agent's individual theoretical lower bound, as defined in Section 5.1) according to Equation 4. As observed in the figure, 79% of the agents improved their performance due to the application of the restructuring heuristic and the performance of 21% of the agents worsened. The average individual improvement, amongst those whose performance improved, is 22% (with a maximum of 55%). The average increase in waiting time overhead, amongst those whose performance worsened is 33% (with a maximum of 93%, equivalent to a 152 increase in the average waiting time). Overall, the average individual relative improvement in the agents' performance (across all agents) is 11%. This moderate improvement is not surprising since the goal of the restructuring method is to improve the system performance rather than the individual performance. For ethical considerations, in order to avoid reducing a student's bonus grade due to the use of input restructuring, we did not include this phase result in the average calculated for those few students who did not benefit from the restructuring. In a way, this is equivalent to applying *balanced* restructuring (see Section 7.3 for details) in order to guarantee that none of the agents lose due to the use of the method.

6.4 Convergence in the Game-Theoretic Stage

Figure 5(b) depicts the average performance of the different agents along the "game-theoretic" part of the experiment, where the use of the game-theoretic approach was proposed to the subjects. We stress that this stage began with a new set of agents, after discussing the game-theoretic solution with the subjects. From the figure, and also evidenced in Table 1, we observe that very few agents actually implemented the suggested game-theoretic approach, and none applied it completely as is, although the implementation of the suggested strategy is very simple (opting-out at $t = 0$ if the agent's processing time is the maximum time and otherwise never opting-out) and was supplied as part of the instructions given. This provides a good indication that the strategy designers did not believe that others will implement that strategy as is. The theoretical result when the agent associated with the maximum processing time is always the first to opt-out at time $t = 0$ is 406 (see Section 5.2). Note that this value is different than the lower bound for the system performance (366) as the suggested strategy is not the optimal one, and was chosen due to its simplicity. In our experiment, the performance of the agents implementing the Nash variants (partial implementation of the suggested Nash strategy) was substantially worse: 533 at the beginning of this stage and 591 at the end of it. The partial implementations of the Nash strategy (denoted 'Nash with Empty Threat') was very quickly abandoned (as evidenced in Table 1) and the steady-state of that stage includes only 2 agents (3%) that actually use the game-theoretic strategy to some extent. It is noteworthy that the difference between the performance of the agents at the beginning of this stage and the performance of the agents at the first stage of the experiment (the "unknown environment") is not statistically significant ($p = 0.28$ for t-test). This strengthens the validity of the Game-Theoretic stage of the experiment, as the first set of agents used in this stage can be considered, once again, agents designed for an unknown environment. Overall, the agents at the steady-state of this stage performed substantially worse in comparison to the steady-state performance with and without the input restructuring heuristic ($p < 0.001$ for t-test).

6.5 Effective Strategy

The primary focus of this research is on the collective system behavior, hence the primary measure used is the average waiting time system-wide. Still, based on the set of agents obtained, complementary agent-level aspects can be investigated. One example of this is the design of a deadlock resolution strategy for an agent operating in environments of the type this paper considers. In this section we suggest a deadlock resolution strategy for the individual agent and demonstrate its effectiveness when applied in the 'unknown' and 'steady-state' environments.

As discussed in Section 6.1, the majority of the agents designed by the strategy designers in our experiment, both for the unknown environment and through the convergence to the steady-state, used strategies whereby the time the agent opts-out positively correlates with the total time and the deadlock size. It is thus a natural choice to use these two parameters when designing an agent to operate in these environments. Our suggested agent design normalizes the two parameters according to their means, and emphasizes each of them according to some power coefficient, resulting in $t = (\frac{Total}{E(Total)})^x * (\frac{Size}{E(Size)})^y$ as the preferred opt-out time. While many strategy designers designed a strategy that determines the opt-out time using a function of the total time and deadlock size, none used a strategy that resembles the one suggested above. Figure 11 depicts the performance of such an agent when it is added to an environment populated with the steady-state set of agents given the values of x and y - the coefficients of the function, and shows that the optimal values are $x = 1.8$ and $y = 3.6$. The graph was generated by iterating over possible values for both x and y, and running 5000 deadlocks for each combination of x and y values, measuring the average waiting time of the proposed agent. For the unknown environment, the graph is very similar and the same values for both x and y coefficients yield the optimal performance.

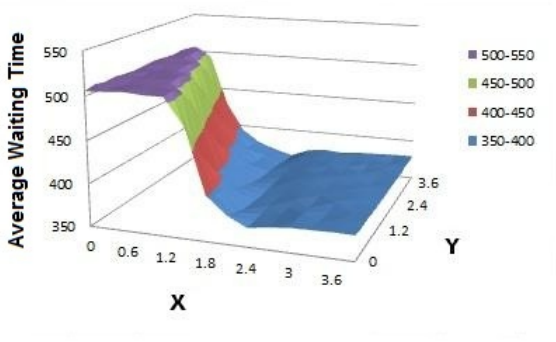

Fig. 11 Single agent performance as a function of the coefficients x and y, when operating with the set of agents used in the steady-state environment.

To analyze the effect of the new agent on the other agents in the system, we ran a system evaluation with and without the agent, both with the set of agents representing the unknown environment and a set representing the steady-state environment (5000 deadlocks for each environment), measuring the average waiting time of each agent of the two sets. In the unknown environment, the inclusion of our proposed agent caused a minor performance degradation to 29 agents with an average of 2.8% (maximum degradation of 5.8%) while 18 agents improved their performance with an average of 2.7% (maximum improvement

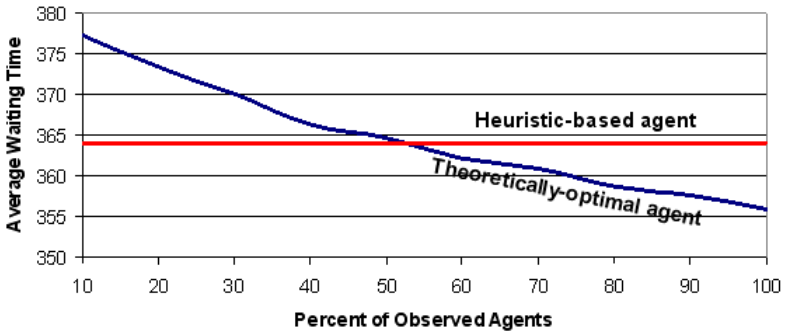

Fig. 12 The average waiting time of the theoretically optimal agent given the percent of agents based on which the function $F_{A_i}(t)$ was generated for each deadlock. The figure also depicts the performance of the heuristic-based agent. When the number of agents observed for generating the function drops below 50%, the theoretically-optimal agent performs worse than the heuristic-based agent.

of 5.4%). In the steady-state environment, the proposed agent caused a minor performance degradation to 24 agents with an average of 2.7% (maximum degradation of 5.1%) while 23 agents improved their performance with an average of 1.9% (maximum improvement of 4%). These results were quite expected, since adding one agent to a system with 66 agents would only have a minor effect on the system. The fact that the individual effect experienced by each agent due to the inclusion of our proposed agent is minor is important since this means that its presence would most probably not drive the strategy designers to change their agent's strategy (i.e., the system is likely to converge to the same steady-state with the suggested agent included).

To better evaluate the effectiveness of our proposed heuristic-based agent, a lower bound for such agent's expected performance was required. For this purpose, a theoretically-optimal agent that acts according to the principles outlined in Section 3 (and in particular in Equation 1) was implemented. This agent was actually given the function $F_{(A,i)}(t)$ (the percentage of agents that opt-out at time t or below when being the i-th agent in deadlock A). The function $F_{(A,i)}(t)$ was generated by replicating the deadlock, however each time with a different agent from the set consisting that state ("unknown" or "steady-state") placed at position i. All the remaining agents in the deadlock were dummy agents that kept holding their resource indefinitely. The deadlock was thus solved by having the agent at position i opt-out eventually. The value of $F_{(A,i)}(t)$ is thus the percentage of instances where the agent at position i opted out before or at time t.[17] This was repeated for all the applicable i values. Based on the function $F_{(A,i)}(t)$ that was generated for each i value, the theoretically-optimal agent was able to fine-grain its opt-out time for the current deadlock it encounters.

The theoretically-optimal agent had an average waiting time of 356, compared to 364 of the heuristic-based agent. The heuristic-based agent is thus highly effective, as it performs only slightly worse than the theoretically-optimal agent, however it does not require a large database of experiences. In fact, to demonstrate the dependency of the theoretically-optimal agent's performance in having perfect knowledge of the function $F_{(A,i)}(t)$ a complementary

[17] If all agents had used a deterministic strategy then only 66 deadlocks would need to be executed this way in order to accurately obtain $F_{(A,i)}(t)$. Nevertheless, as indicated in Table 1, one of the agents used some randomness in its strategy. Therefore for that agent a thousand deadlocks were generated, capturing the distribution of its opt-out time. These were merged with the remaining 65 observations, weighted accordingly.

experiment was conducted. In this experiment, instead of enabling the generation of the function $F_{(A,i)}(t)$ based on all 66 agents in each of the different possible positions, only a portion of the agents was used for each position. Figure 12 depicts the performance of the theoretically-optimal agent as a function of the percent of agents that are used for generating the function $F_{(A,i)}(t)$. As observed in the figure, if less than 50% of the agents are used, the performance of the theoretically-optimal agent becomes worse than the performance of the proposed heuristic-based agent, emphasizing, once again, the effectiveness of the latter. In addition, we note that while in our experiments the database for generating $F_{(A,i)}(t)$ was built in an ad-hoc manner, based on the current deadlock, in reality this database would need to be generated offline (assuming the other agents are available for generating such a database). Obviously this is impractical, given the numerous possible deadlocks the agent may run into when executing its task.

6.6 Machine Learning Approach

The performance of the input restructuring approach in the distributed deadlock resolution domain is principally dependent on the ability to predict the agents' behaviors in different deadlocks. This ability enables the restructuring algorithm to influence the agents to act in a way that the preferred agent according to Equation 2 opts-out as early as possible while the other agents opt-out at any time after that agent. For this purpose, our heuristic uses a restructured deadlock that is based on one of the formerly encountered deadlocks, assuming that with a high probability, the agent's opt-out time will be similar to what it was in the previously encountered non-restructured deadlock. A possible alternative to this approach is the use of legacy machine learning methods, with the premise to approximate functions based on sample inputs and outputs of that function, which can be found in the literature (an agent strategy can be modeled as a function that receives the processing times as inputs, and outputs the opt-out time t). In this section we outline the performance of our proposed heuristic in comparison to one of the legacy machine learning methods: neural networks. For this purpose, we have augmented the heuristic described in Algorithm 1, based on a multi-layered, feed-forward neural network. The new addition and is desinged to replace the learning and exploitation parts of our heuristic. The multi-layered, feed-forward neural network, also referred to as a multi-layered perceptron, with a sigmoid activation function was originally proven to be a universal function approximator [13]. Since then, much research has been carried out on the applications of neural networks for function approximation and behavior prediction in many domains [43, 20] and hence the choice of neural networks as a machine learning algorithm for learning agents' behaviors as a means of prediction is a natural choice in our case.

The new implementation is based on having the system keep a neural network NN_i for each agent A_i, which is used to predict its assigned agent's opt-out time given any arbitrary deadlock (NN_i is trained to approximate the function $(Size, Total, MyTime) \rightarrow t$, which models the agent's strategy, where t is the time at which NN_i predicts that agent A_i is going to opt-out given the deadlock parameters). The system trains each neural network yet again every time an additional set of c deadlocks in which A_i has opted-out is added to the database (were c is a constant training interval). The system only considers deadlocks in which A_i opted-out rather than any deadlock A_i was in, as only in those deadlocks the actual opt-out time is known and stored in the database. This value is used as the expected output for the supervised training of NN_i. The training dataset's size increases with time,

as the agents experience new deadlocks. This enables NN_i to predict A_i's behavior more accurately but also requires more time for the training process.

Once the system has a trained NN_i, it can use it to choose a deadlock to be presented to agent A_i when it is chosen for exploitation. The process of choosing a deadlock in this latter case is based on randomly generating deadlocks (based on the different parameters' distributions), in which the processing time of A_i equals its actual processing time in the current deadlock. The neural network NN_i is then used to predict the opt-out time for each generated deadlock. The deadlock that is eventually presented to the agent is the one that best fits the design goals outlined in Equation 2 (either the deadlock in which the agent is expected to opt-out the latest, or the earliest).

Algorithm 2 Neural-Network-Based Exploitation (replacing steps 10-15 in Algorithm 1)

1: **for** $A_i \in Exploit$ **do**
2: Set $MinA_i = null, PredictedTime_{MinA_i} = null, MaxA_i = null, PredictedTime_{MaxA_i} = null$
// Generate a total of $RandomDeadlocks$ random deadlocks according to the parameters distributions (with the actual processing time of agent A_i) and evaluate them
3: **for** $j = 1$ To $RandomDeadlocks$ **do**
4: Set *d*=RandomDeadlock()
// Use the neural network assigned to agent A_i to predict its opt-out time for d, according to the deadlock size, the total processing time of all agents in the deadlock and agent A_i's own processing time
5: Set $predictedTime = NN_i.predict(d.size, d.totalTime, d[A_i].processingTime)$
6: **if** $predictedTime < PredictedTime_{MinA_i}$ **then**
7: Set $PredictedTime_{MinA_i} = predictedTime$
8: Set $MinA_i = d$
9: **else if** $predictedTime > PredictedTime_{MaxA_i}$ **then**
10: Set $PredictedTime_{MaxA_i} = predictedTime$
11: Set $MaxA_i = d$
12: **end if**
13: **end for**
// Calculate the expected performance when agent A_i opts-out first, given the expected resolution time in its minimal deadlock $MinA_i$
14: Set $Performance_i = \left(\sum_{j \neq i} \sum_{A_k \in A_{sub}^{current}(A_j, A_i)} t_{A_k} + \sum_{k \neq i} t_{A_k}\right) + PredictedTime_{MinA_i}$
15: **end for**

The use of the neural-network-based deadlock selection is described in pseudo code in Algorithm 2. For clarity, Algorithm 2 specifies only the part that needs to be replaced in Algorithm 1 (steps 10-15), i.e., the learning of the agents' behaviors and exploiting that knowledge. Unlike Algorithm 1 where the previously encountered maximal and minimal deadlocks of an agent are restructured in order to control its behavior, the neural-network-based solution generates a set of random deadlocks ($RandomDeadlocks$) for every agent $A_i \in Exploit$. The value of $RandomDeadlocks$ is part of the input and defines the number of new deadlocks that will be generated and evaluated for each agent that is planned to be exploited. Any increase in $RandomDeadlocks$ will further improve the chance of finding a more preferable deadlock for the agent, however it will also require investing more computational resources in the evaluation of the additional deadlocks. The randomly generated deadlocks are generated according to the parameters distributions used by the system, except for the processing time of the agent that is exploited, which is set to its actual processing time in the current deadlock (Step 4). The algorithm then uses NN_i to predict the

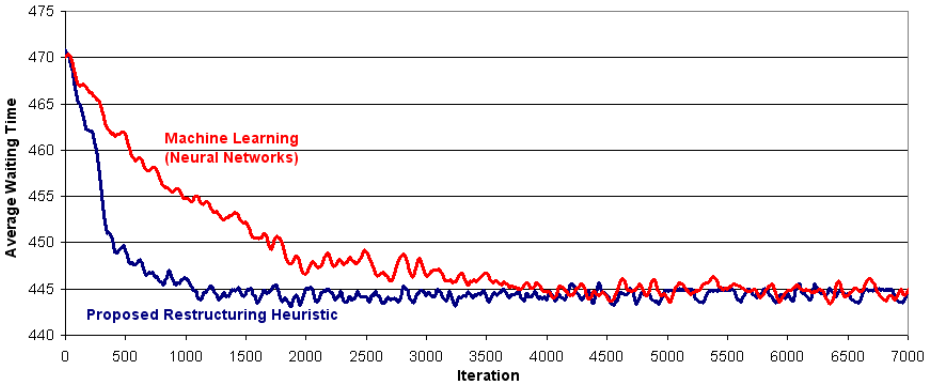

Fig. 13 Average waiting time (system performance) as a function of the deadlocks observed by the system (in comparison to the neural-network-based restructuring heuristic). Both algorithms reach an almost identical performance after convergence, however the proposed restructuring heuristic converges substantially faster.

time the agent is expected to opt-out when in the current randomly generated deadlock d (Step 5). Finally, $MinA_i$ and $MaxA_i$ are selected according to the predicted opt-out times for each randomly generated deadlock (Steps 6-12). While the learning part of the proposed restructuring heuristic consists only of adding the deadlock result to the database when the opting-out agent is explored, the neural-network-based solution requires an additional step of training the neural network of any agent A_i, every time a new set of c deadlocks in which A_i opted-out first are added to the database.[18]

The implementation of the neural-network-based solution was based on the Neuroph open source library [29]. Each network was implemented using a multi-layered perceptron with the backpropogation learning algorithm [61], and had one hidden layer with 20 neurons and a sigmoid activation function. This perceptron architecture yielded the best predictive accuracy based on an empirical comparison of different architectures, ranging from 1 to 3 hidden layers and 1-100 neurons per hidden layer. Each neural network A_i was trained every time additional 10 deadlocks in which agent A_i opted-out first were added to the database (i.e., c=10). Using a smaller learning interval would improve intermediate results, but would not improve the overall convergence pattern nor the performance after convergence. The number of random deadlocks that were generated and evaluated for each agent as part of the exploitation phase was 1000. Note that the typical number of deadlocks in DB_i, which is the number of observations that were available to the proposed restructuring heuristic for learning agent A_i's behavior, is much smaller than 1000 and thus our experimental setting gives the neural-network-based solution an advantage of a larger pool of deadlocks that could be used for exploitation.

Figure 13 comparatively depicts the average waiting time (system performance) as a function of the number of deadlocks observed by the system (each data point is the average of 1000 simulations with the 66 agents). From Figure 13 we observe that although the neural-network-based solution performs almost as good as the proposed restructuring heuristic once it converges, it requires substantially more observations to converge (about 5 times the observations the proposed restructuring heuristic requires). This is because the neural networks require a large amount of training samples to accurately learn to predict

[18] The use of $c = 1$ suggests continuous training, however requires investing substantial computational power for training on each deadlock.

agents' behaviors, while the proposed restructuring heuristic manages to produce qualitative prediction with substantially less observations due to the use of a more domain-specific approach. Another advantage of using the proposed restructuring heuristic over the machine-learning-based algorithm, which naturally cannot be reflected in the figure, is the amount of resources the neural networks training process requires. While running 1000 simulations with the proposed restructuring heuristic takes only a few minutes, running the same amount of simulations with the machine-learning-based algorithm takes about 3 days, when most of the time is spent on the neural networks' training.

7 Complementary Experimentation

In this section we report the results of three complementary experiments, aiming to highlight some additional aspects of the distributed deadlock resolution through the input restructuring approach. The first attempts to identify the source of improvement, distinguishing between the deadlock resolution time and the choice of the agent to opt-out first. The second aims to investigate the usefulness of supplying the agents deadlock-related information in the first place. The third, acknowledges the fact that the improvement in system performance is also accompanied by a decrease in some of the agents' individual performance. It therefore evaluates the ability to apply some balancing, favoring agents that were first to opt-out in prior deadlocks.

7.1 Source of Improvement

We note that the system's average waiting time is affected by two factors. The first is the time at which deadlocks are resolved. The second is the choice of the agent resolving the deadlock (and in particular, its characteristics, such as its position in the deadlock and processing time). In order to test our hypothesis that the latter factor is more crucial to the performance of the system, we designed an experiment in which each deadlock was injected with a dummy agent in a random position, that always opts out at time $t = 0$. This simulates an environment in which each deadlock is always resolved immediately, and the agent resolving it is in a random position within the deadlock. Note that the goal of the restructuring heuristic is to optimize the system performance by controlling the agent that resolves each deadlock rather than to optimize the resolution time of each deadlock (the restructuring heuristic attempts to influence the agent that minimizes the average waiting time (according to Equation 2) to opt-out as quickly as possible, but does not always succeed in influencing it to opt-out at time $t = 0$). It is also noteworthy that in this experiment, the strategies of the other agents are irrelevant, as the deadlocks are always resolved by the injected agents at time $t = 0$. The average waiting time of the system in this experiment, obtained as an average of 5000 simulated deadlocks, was 506, compared to 446 in the restructured steady-state environment, which reflects a substantial difference of 57% in the system performance. We thus conclude that while the resolution time of deadlocks affects the system performance, the dominant factor is the choice of the agent that resolves each deadlock.

7.2 The Usefulness of Information Disclosure

The underlying assumption of the deadlock model this paper considers is that subjects receive the deadlock description as input. While the results achieved with the restructuring

heuristic for this case are encouraging, one may wonder if the idea of supplying the agents the deadlock description in the first place is useful. It is possible that without the deadlock parameters the system's steady-state performance would have been initially better than in our case. For this purpose, a complementary experiment was conducted, in an attempt to evaluate the steady-state system performance when agents do not receive any information about the current deadlock as input. In order to avoid a carryover effect, a separate set of 25 subjects participated in the experiment. The subjects were given the exact same instructions with all the details relating to the parameters' distributions. This time, however, the subjects were merely requested to supply their opt-out time, with no knowledge about the current deadlock they are in. In this experiment we followed the same methodology of converging to steady-state, and thus the subjects were allowed to change their strategy as many times as they desired while the system and individual evaluations took place.

The average waiting time in the resulting steady-state in this experiment was 630, compared to 475 reported for the steady-state in the case where agents receive the deadlock description as input (statistically significant for $p < 0.02$ in t-test). In comparison to the restructuring based steady-state, the difference in performance is substantially greater (statistically significant for $p < 0.001$). Restructuring the input in the case where no information is supplied to the agents is, of course, inapplicable. We thus conclude that the approach of disclosing information about formed deadlocks, such as their size and processing times of the different agents in the deadlock, dominates the approach of not supplying any information other than notifying the agents that the system is in a deadlock.

7.3 Balancing Between System and Individual Performance

Overall, the restructuring heuristic is designed to resolve the deadlock in a way that the total waiting time is minimized, by attempting to increase the probability of having the agents, which, by opting-out, yield the minimal average waiting times (system-wise) according to $Performance_i$ (Step 14 in Algorithm 1), indeed opt-out first. This is achieved by urging these agents to opt-out as early as possible and at the same time, trying to influence the other exploited agents to opt-out as late as possible. While this is beneficial system-wise, it may result in a decrease in the performance of some individual agents. Therefore, for cases where the system designer prefers some tradeoff between system performance and individual performance, a more balanced restructuring heuristic may be considered. In its balancing version, the heuristic keeps track of agents that were more affected by the restructuring that took place until then (i.e., ended up opting-out more frequently than other agents) in former deadlocks by managing a counter C_i that is incremented whenever an agent opts-out first after it is presented with a restructured deadlock. The agents that were more affected by the restructuring heuristic, will now be assigned a decreased exploitation probability in future deadlocks (this can be done by modifying Step 4 of Algorithm 1 in a way that increases A_i's exploration probability the bigger C_i is).

The extent of using balancing can be controlled by assigning a probability P_b for choosing whether to apply balancing, which means using the C_i values for increasing the exploration probability for all exploited agents in the deadlock, or ignoring the C_i values for all exploited agents in the deadlock (this can be achieved by adding a step before Step 3, where, with a probability of P_b, we choose whether or not to apply balancing to all exploited agents in the current deadlock). The greater the value of P_b, the lower the improvement achieved in system performance. This is due to the fact that the balancing restructuring heuristic will try to balance its effects on the agents more often, in an attempt to influence agents to opt-

out first sub-optimally, which does not align with the desirable behavior system-wise (e.g., when the best agent to opt-out according to Equation 2 has already been exploited many times compared to the other agents, the heuristic tries to influence another agent to opt-out first although this is sub-optimal system-wise). For similar reasons, the greater the value of P_b the smaller the degradation in individual performance.

The use of balanced restructuring enables achieving an all-win result, where all agents improve their expected individual performance, however with the price of a substantially lower overall system improvement. Still, controlling the value of P_b, a substantial system improvement can be achieved while dramatically decreasing the individual loss due to the restructuring: for example, in our experiments with balancing we managed to balance the loss such that, although the number of agents experiencing loss slightly increased, the average and maximum individual loss was substantially decreased. In particular, we managed to achieve 55% of the system-wise improvement reported in the former section (i.e., the improvement compared to steady-state with no restructuring) while reducing the average individual performance reduction for those agents that suffer from the restructuring by 67%, and reduce the maximum individual loss by 63%.

8 Related Work

Relevant literature is cited wherever applicable throughout the paper. In this section we introduce additional work that is relevant to the methodology used in this paper, and emphasize the differences between these works and ours.

Deadlock management has been studied in negotiation, tasks assignment and various other research domains [18,62,57]. The majority of deadlock research is associated with Operating Systems [55,52,2,42] as it is a fundamental issue in that area. The main strategies used for dealing with deadlocks include simply ignoring the deadlock problem altogether (the Ostrich approach), detecting deadlocks and when they occur, taking steps to recover (Detection and Recovery) using a centralized mechanism [36,35] or using a distributed protocol [41,55,51] to avoid deadlocks by carefully allocating resources or negating the conditions for their occurrence (Deadlock Avoidance) [37,6,32]. Yet, as discussed earlier in the paper, most of the research in this area assumes that agents are inherently cooperative and therefore the solutions proposed are preemption-based [37,6,32] or based on pre-defined protocols for determining the agent to opt-out to which all agents consent and act accordingly [36,51,41,35,55].[19] In the type of systems this paper considers, in wherein the agents are self-interested and bounded-rational, however, agents would only act according to a preferred protocol as long as they believe the other agents deem that the protocol serves their interests (and even if the protocol does serve their interest, the agents would still have to be convinced of this fact because of their bounded-rational nature). To the best of our knowledge, no prior work has considered the deadlock resolution problem in a fully distributed setting with self-interested bounded-rational agents as considered in this paper.

As discussed in the preceding sections, an optimal solution to the problem may be designed using game theory principles. While game-theoretic approaches are widely used in studying conflict situations, there is extensive evidence in literature of the failure of such approaches in settings where the main players are people or bounded-rational agents [4,40, 64,7]. People are known to be bounded-rational [53], cannot be trusted to exhibit optimal

[19] Same holds for multi-agent settings in general, where agents are commonly fully cooperative or have the same goal [24].

behavior [46] and often tend not to use the optimal strategy even when one is provided [33]. On the other hand, works that report the successful use of game-theoretic approaches, in particular in repeated interaction domains [39,15,1,45,22] can also be found. Therefore, the success of such approaches in the deadlocks domain is a priori inconclusive and thus the experimental evidence given in this paper as to the failure of the approach in this domain is of great significance.

Diversified work has addressed the challenge of improving people's decision-making, mainly by developing decision support systems to assist users in gathering, merging, analyzing, and using information to assess risks and make recommendations in situations that may require a tremendous amount of the users' time and attention [65,17]. Recently, several approaches have been proposed that attempt to reconstruct the decision-making problem [58,50,4,31] instead of attempting to change people's decision-making strategies directly. None of the prior work, however, involves strategic settings where an agent's strategy is influenced by the strategies of others, as well as evolving settings. Furthermore, all these work considered a single decision maker instead of a multi-party setting, in which the collective behavior should be optimized and thus are irrelevant to the settings considered in this paper.

The use of agents programmed by people to test multi-agent mechanisms has become quite common in recent years [40,9]. Much effort in multi-agent systems research has been dedicated to examining people's use of agents designed to represent them and act on their behalf [8,48,49,40,38]. For example, Kasba [11] is a virtual marketplace on the Web where people create autonomous agents in order to buy and sell goods on their behalf. Various research has involved programming agents in the decision-theoretic framework of the Colored-Trails game [23]. In this game the agents had to reason about other agents' personalities in environments in which agents are uncertain about each other's resources. Unlike this line of work, where the platform is used for enabling people's representation by automatic agents, our work uses agents programmed by people as bounded-rational agents in general, and focuses on developing a means of actively affecting their decisions in order to improve the overall system performance.

9 Discussion, Conclusions and Future Work

The deadlock problem is inherent in physical and virtual environments, whenever resources are limited and decision makers are self-interested. The problem is challenging for the most part when the decision makers are bounded-rational (or pre-programmed with different deadlock handling logics, as in our case). Here, the many deadlock management solutions that were designed in prior literature either for centrally-managed or fully distributed systems are not effective, as they all rely on the agents' cooperation whereas in our case the system designer cannot enforce the desired collective behavior. Instead, each individual agent needs to be incentivized to comply with the behavior desired by the system designer. To the best of our knowledge, we are the first to address the distributed deadlock management problem with bounded-rational self-interested agents.

One possible solution discussed in the paper for the deadlock problem is game-theoretic-based. Its importance is in its potential to achieve optimal performance, as such a solution, if successfully applied, guarantees the best system performance even though the agents are not fully cooperative (i.e., each attempts to minimize its own expected waiting time). The evidence given in our experiments of the failure of the game-theoretic-based approach is not surprising. Earlier literature from other domains has often shown that such solutions do not hold whenever people or rationality bounded agents are concerned. Still, it is interesting to

see that none of the participants in our experiments actually implemented the game-theoretic equilibrium strategy, despite the fact that it was practically given to them and its principles were thoroughly explained.

The experimental design used in this paper places strong emphasis on convergence to a steady state. Throughout the experiments, the system converged to a steady state four times (from an unknown environment to a steady-state, from a steady-state to a restructuring-based steady-state, in the game-theoretic stage and in the complementary experiment reported in Section 7.2). It is noted that such convergence cannot be taken for granted, as discussed in the related work section. The importance of having the system converge to a steady state is primarily in strengthening the significance of the results achieved, as the improvement is measured at a point where the strategies are not likely to further change. In particular, it is important when considering the improvement achieved with the restructuring heuristic, as it suggests that the improvement reported is not momentary but one that will last.

The uniqueness of the restructuring heuristic used in this paper, in comparison to other approaches for deadlock management reported in the literature, is that it does not explicitly require the agents' cooperation. Instead, it indirectly influences the agents' deadlock-related decisions in a way that improves the overall system performance. This saves the trouble of constructing the appropriate incentives for the self-interested agents, needed in order to gain their cooperation, and eliminates the need to use argumentation, to convince the agents as to the benefits of the suggested cooperation (in case they do not realize it due to their bounded rationality or limited computational capabilities). It is also very useful whenever the agents' strategy is pre-set and cannot be changed externally, unless it is re-programmed. In such cases, it is the only way available to influence the decisions the agents make. Despite its many advantages, the use of input restructuring needs to be carefully handled, since agents may not always react to the restructured information the way we would expect them to. In particular, in our case, unlike in prior attempts to use input restructuring in other domains, the idea is to affect the collective behavior of a group of agents rather than a single agent, by supplying a revised input to each agent separately. The risk in doing so is that the resulting change in the performance of any individual agent in the system, either due to restructuring its own input or due to changes in the behaviors of others, may cause that agent's designer/owner to re-design and make further changes in its strategy and the system may ends up doing worse than with the original agents' strategies. In this sense, the results reported in this paper are highly encouraging as they show that with the restructuring heuristic the system converges to a steady state in which the system performance as well as individual performance substantially improves.

Alongside its many advantages, the restructuring approach presented in this paper has some limitations. First, the applicability of the approach depends on the ability to identify the agents. This capability typically exists in real systems, by means of cookies, as well as various other enforceable identification methods, as discussed in Section 4. Second, the performance of the restructuring approach highly depends on the size of the database of prior experiences of the agents that can be utilized for generating the restructured inputs. Our experiment, comparing the method with neural-network-based implementations, demonstrates the efficiency of the proposed solution in the sense that the suggested heuristic produces restructured inputs that effectively affect the agents using a substantially smaller amount of such data. Finally, as in all prior work that considers problem restructuring, the fact that the input received is restructured (i.e., the deadlock presented may be different in large from the real one) should not be disclosed to the agents and their strategy designers, as the idea is that they will act according to what they believe is the best response to the input they receive. The question of whether or not providing restructured information can be justified

if it serves a good purpose (e.g., tell someone we go to dinner when we are actually taking him to a surprise party thrown to him or tell a child that the tooth fairy exists to protect her innocence), is beyond the scope of this paper. The paper merely establishes that whenever such a requirement is feasible, the improvement achieved with the proposed method (both system-wise and individually), as demonstrated in this paper, is substantial. Furthermore, with the use of balancing, as reported in 7.3, we can guarantee that none of the agents will suffer individually when applying this approach. We believe this is feasible in open systems where there is no binding commitment of the hosting operating system of any kind.

Overall, the restructuring approach used in the paper can potentially be used in any of the domains where prior work has considered input restructuring for a single agent (see references throughout the paper). It is particularly useful in strategic settings whenever the goal is to affect the collective behavior of a group of agents. Furthermore, there is much work on influencing the decision of a group of self-interested agents that must reach an agreement on a joint action. For example, convincing a group of people who share an office to agree on an economy mode of the air-conditioning and low light intensity (e.g., [26]). The agents in this line of work are taken to be self-interested but taking the agents to be people or bounded-rational agents is as much realistic and in such setting the methods developed in our paper can be of much use.

There are several directions that we would like to explore for future research based on the work reported in this paper. The first is the possibility of improving the restructuring heuristic by integrating a classifier in it to classify agents according to their deadlock management decisions over time. Such classification could be then used to augment the set of deadlock experiences of an agent by adding experiences of other agents of the same class. Another possible extension is to improve the heuristic-based agent by taking into account other deadlock related parameters in addition to the total time and size parameters, such as the distribution of the processing times.

Finally, we note that the model used in this paper considers a quite standard deadlock setting (e.g., a cyclic deadlock). While this typically characterizes most deadlocks occurring in operating systems, multi-agent systems in general are more complex and agents have plenty of different resources, alternatives and actions. The choice of the deadlock model made in this research was motivated primarily by the tradeoff between the complexity and richness of the deadlock model and the overhead and utilization of the resources available for the experiments as well as the ability to properly control the different affecting parameters. We believe that many of the ideas given in the paper carry over to more complex MAS settings, with the most apparent change being that a larger database of prior experiences is required to effectively predict the influence of different problem setting over individual agents' decisions. As such, we see much room for applying our approach and experimenting with more complex deadlock settings, e.g., deadlocks that involve several resources held by each agent or settings where an agent may be involved in more than a single deadlock at the same time.

Acknowledgment

Our results concerning the failure of the game-theoretic approach in the distributed deadlock resolution domain appear in the proceedings of AAMAS 2012 [54].

References

1. D. Abreu and A. Rubinstein. The structure of nash equilibrium in repeated games with finite automata. *Econometrica*, 56(6):1259–1281, 1988.
2. R. Agrawal, M. Carey, and L. Mcvoy. The performance of alternative strategies for dealing with deadlocks in database management systems. *IEEE Transactions on Software Engineering*, 13(12):1348–1363, 1987.
3. P. Auer, N. Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
4. A. Azaria, Z. Rabinovich, S. Kraus, and C. Goldman. Strategic information disclosure to people with multiple alternatives. In *proceedings of AAAI*, pages 594–600, 2011.
5. R. Azoulay-Schwartz, S. Kraus, and J. Wilkenfeld. Exploitation vs. exploration: choosing a supplier in an environment of incomplete information. *International Journal on Decision Support Systems and Electronic Commerce*, 38(1):1–18, 2004.
6. S. Bensalem, J. Fernandez, and K. Havelund. Confirmation of deadlock potentials detected by runtime analysis. In *proceedings of PADTAD*, pages 41–50, 2006.
7. P. Bo and G. Frechette. The evolution of cooperation in infinitely repeated games: Experimental evidence. *American Economic Review*, 101(1):411–429, 2011.
8. M. Chalamish, D. Sarne, and S. Kraus. Programming agents as a means of capturing self-strategy. In *proceedings of AAMAS*, pages 1161–1168, 2008.
9. M. Chalamish, D. Sarne, and R. Lin. Enhancing parking simulations using peer-designed agents. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1–7, 2012.
10. K. Chandy, J. Misra, and L. Haas. Distributed deadlock detection. *ACM Transactions on Computer Systems*, 1(2):144–156, 1983.
11. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *proceedings of PAAM*, pages 75–90, 1996.
12. E. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys*, 3(2):67–78, 1971.
13. G. Cybeko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 5(4):455–455, 1992.
14. L. Cysneiros and E. Yu. Requirements engineering for large-scale multi-agent systems. In *Proceedings of Software Engineering for Large-Scale Multi-Agent Systems*, pages 39–56, 2002.
15. J. Duffy and E. Hopkins. Learning, information and sorting in market entry games: Theory and evidence. ESE Discussion Papers 78, University of Edinburgh, 2004.
16. A. Elmalech and D. Sarne. Evaluating the applicability of peer-designed agents in mechanisms evaluation. *in Proceedings of IAT*, 2012.
17. Y. Elmaliach and G. Kaminka. Robust multi-robot formations under human supervision and control. *Journal of Physical Agents*, 2(1):31, 2008.
18. U. Endriss. Monotonic concession protocols for multilateral negotiation. In *proceedings of AAMAS*, pages 392–399, 2006.
19. I. Erez and A. Roth. Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *American Economic Review*, 88(4):848–881, 1998.
20. S. Ferrari. Smooth function approximation using neural networks. *IEEE Transactions on Neural Networks*, 16(1):24–38, 2005.
21. M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The digital distributed system security architecture. In *proceedings of National Computer Security Conference*, pages 305–319, 1989.
22. P. Gmytrasiewicz and E. Durfee. Rational coordination in multi-agent environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(4):319–350, 2000.
23. B. Grosz, S. Kraus, S. Talman, B. Stossel, and M. Havlin. The influence of social dependencies on decision-making: Initial investigations with a new game. In *proceedings of AAMAS*, pages 780–787, 2004.
24. M. Hadad, S. Kraus, I. B.-A. Hartman, and A. Rosenfeld. Group planning with time constraints. *Annals of Mathematics and Artificial Intelligence*, pages 1–49, 2013.
25. C. Hajaj, N. Hazon, D. Sarne, and A. Elmalech. Search more, disclose less. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.
26. N. Hazon, R. Lin, and S. Kraus. How to change a group's collective decision? In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 198–205. AAAI Press, 2013.
27. K. Hirayama and J. Toyoda. Forming coalitions for breaking deadlocks. In *proceedings of AAAI*, pages 155–162, 1995.

28. M. Hornick and S. Zdonik. A shared, segmented memory system for an object-oriented database. *ACM Transactions on Information Systems*, 5(1):70–95, 1987.
29. http://neuroph.sourceforge.net.
30. S. Isloor and T. Marsland. The deadlock problem: An overview. *IEEE Computer*, 13(9):58–78, 1980.
31. S. Iyengar. *The Art of Choosing*. Twelve, 2010.
32. M. Jager and B. Nebel. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *proceedings of IEEE Intelligent Robots and Systems*, pages 1213 – 1219, 2001.
33. D. Kahneman and A. Tversky. *Choices, values, and frames*. Cambridge University Press, 2000.
34. N. Kaveh and W. Emmerich. Deadlock detection in distributed object systems. In *proceedings of ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 44–51, 2001.
35. A. Kshemkalyani and M. Singhal. Efficient detection and resolution of generalized distributed deadlocks. *IEEE Transactions on Software Engineering*, 20(1):43–54, 1994.
36. S. Lee. Fast, centralized detection and resolution of distributed deadlocks in the generalized model. *IEEE Transactions on Software Engineering*, 30(9):561–573, 2004.
37. P. Li, K. Agrawal, J. Buhler, R. Chamberlain, and J. Lancaster. Deadlock-avoidance for streaming applications with split-join structure: Two case studies. In *proceedings of IEEE Application-specific Systems Architectures and Processors*, pages 333–336, 2010.
38. R. Lin, S. Kraus, Y. Oshrat, and Y. Gal. Facilitating the evaluation of automated negotiators using peer designed agents. In *proceedings of AAAI*, pages 817–822, 2010.
39. G. Mailath. Do people play nash equilibrium? lessons from evolutionary game theory. *Journal of Economic Literature*, 36(3):1347–1374, 1998.
40. E. Manisterski, R. Lin, and S. Kraus. Understanding how people design trading agents over time. In *proceedings of AAMAS*, pages 1593–1596, 2008.
41. D. Mitchell and M. Merritt. Distributed algorithm for deadlock detection and resolution. In *proceedings of ACM Symposium on Principles of Distributed Computing*, pages 282–284, 1984.
42. C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the r* distributed database management system. *ACM Transactions on Database Systems*, 11(4):378–396, 1986.
43. K. Narendra. Adaptive control using neural networks and approximate models. *IEEE Transactions on Neural Networks*, 8(3):475–485, 1997.
44. T. Nguyen, M. Roos, and J. Rothe. A survey of approximability and inapproximability results for social welfare optimization in multiagent resource allocation. *Annals of Mathematics and Artificial Intelligence*, pages 1–26, 2013.
45. M. Parameswaran, H. Rui, and S. Sayin. A game theoretic model and empirical analysis of spammer strategies. In *proceedings of Collaboration, Electronic Messaging, AntiAbuse and Spam*, 2010.
46. M. Rabin. Psychology and economics. *Journal of Economic Literature*, 36(1):11–46, 1998.
47. M. Roesler and W. Burkhard. Resolution of deadlocks in object-oriented distributed systems. *IEEE Transactions on Computers*, 38(8):1212–1224, 1989.
48. A. Rosenfeld and S. Kraus. Modeling agents through bounded rationality theories. In *proceedings of IJCAI*, pages 264–271, 2009.
49. A. Rosenfeld and S. Kraus. Modeling agents based on aspiration adaptation theory. *Journal of Autonomous Agents and Multi-Agent Systems*, 24(2):221–254, 2012.
50. D. Sarne, A. Elmalech, B. Grosz, and M. Geva. Less is more: Restructuring decisions to improve agent search. In *proceedings of AAMAS*, pages 431–438, 2011.
51. S. Selvaraj and R. Ramasamy. An efficient detection and resolution of generalized deadlocks in distributed systems. *International Journal of Computer Applications*, 1(1):1–7, 2010.
52. A. Silberschatz, G. Gagne, and P. Galvin. *Operating System Concepts*. John Wiley & Sons, 8th edition, 2008.
53. A. Simon. Theories of bounded rationality. In C. B. McGuire and R. Radner, editors, *Decision and Organization*, pages 161–176. Amsterdam: North Holland, 1972.
54. N. Sofy and D. Sarne. On the failure of game theoretic approach for distributed deadlock resolution. In *Proceedings of AAMAS*, pages 1445–1446, 2012.
55. S. Srinivasan and R. Rajaram. A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems. *Distributed and Parallel Databases*, 29(4):261–276, 2011.
56. W. Stirling, M. Goodrich, and D. Packard. Satisficing equilibria: A non-classical theory of games and decisions. *Journal of Autonomous Agents and Multi-Agent Systems*, 5(3):305–328, 2002.
57. P. Sujit, A. Sinha, and D. Ghose. Multiple uav task allocation using negotiation. In *proceedings of AAMAS*, pages 471–478, 2006.
58. R. Thaler and C. Sunstein. *Nudge: Improving decisions about health, wealth, and happiness*. Yale University Press, 2008.

59. A. Tversky and D. Kahneman. The framing of decisions and the psychology of choice. *Science*, 211(4481):453–458, 1981.
60. J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *proceedings of European Conference on Machine Learning*, pages 437–448, 2005.
61. P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
62. D. Weyns, N. Boucke, and T. Holvoet. A field-based versus a protocol-based approach for adaptive task assignment. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):288–319, 2008.
63. N. Wijngaards, B. Overeinder, M. V. Steen, and F. Brazier. Supporting internet-scale multi-agent systems. *Data and Knowledge Engineering*, 41:229–245, 2002.
64. J. Wright and K. Brown. Beyond equilibrium: Predicting human behavior in normal-form games. In *proceedings of AAAI*, pages 901–907, 2010.
65. E. Yu. Evolving and messaging decision-making agents. In *proceedings of AGENTS*, pages 449–456, 2001.